எளிய தமிழில் Deep Learning

[Machine Learning - பாகம் 2]

து.நித்யா

எளிய தமிழில் Deep Learning

(Machine Learning - பாகம் 2)

து. நித்யா

nithyadurai87@gmail.co
m

மின்னூல் வெளியீடு :

கணியம் அறக்கட்டளை

kaniyam.com

அட்டைப்படம் - லெனின் குருசாமி,
guruleninn@gmail.com

மின்னூலாக்கம் : த. சீனிவாசன்
tshrinivasan@gmail.com

முதல் பதிப்பு மார்ச் 2020

ஆழக் கற்றல் - Deep Learning – கணினி உலகில் அதி வேகமாக வளர்ந்து வரும் துறை Machine Learning ன் உயர்நிலைத் துறை ஆகும். இதை, இந்த நூல் எளிமையாக அறிமுகம் செய்கிறது.

தமிழில் கட்டற்ற மென்பொருட்கள் பற்றிய தகவல்களை “கணியம்” மின் மாத இதழ், 2012 முதல் வெளியிட்டு வருகிறது. இதில் வெளியான Deep Learning பற்றிய கட்டுரைககளை இணைத்து ஒரு முழு புத்தகமாக வெளியிடுவதில் பெரு மகிழ்ச்சி கொள்கிறோம்.

உங்கள் கருத்துகளையும், பிழை திருத்தங்களையும் editor@kaniyam.com க்கு மின்னஞ்சல் அனுப்பலாம்.

http://kaniyam.com/learn-deep-learning-in-tamil என்ற முகவரியில் இருந்து இந்த நூலை பதிவிறக்கம் செய்யலாம். உங்கள் கருத்துகளையும் இங்கே பகிரலாம்.

படித்து பயன் பெறவும், பிறருடன் பகிர்ந்து மகிழவும் வேண்டுகிறோம்.

கணியம் இதழை தொடர்ந்து வளர்க்கும் அனைத்து அன்பர்களுக்கும் எமது நன்றிகள்.

த.சீனிவாசன்

tshrinivasan@gmail.com

ஆசிரியர்
கணியம்

editor@kaniyam.com

பொருளடக்கம்

உரிமை

நூல் மூலம் :

http://static.kaniyam.com/ebooks/learn-deep-learning-in-tamil.odt

ஆசிரியர் உரை:

“எண்ணித் துணிக கருமம் துணிந்தபின்

எண்ணுவோம் என்பது இழுக்கு ”

என்ற திருக்குறளின் படி, deep learning பற்றி ஒரு புத்தகத்தை எழுதிவிடலாம் என்று துணிந்து எண்ணிவிட்டேன். ஆனால் எழுதத் தொடங்கிய பின்புதான் தெரிகிறது இது ஒரு மாபெரும் கடல் என்று! இந்தக் கடலில் மூழ்கி முத்தான ஒரு புத்தகத்தைப் படைக்க வேண்டும் என்றால் எனக்குப் பல வருடங்கள் பிடிக்கும். ஆகவே என்னால் முடிந்த வரை deep learning -ன் கீழ் வரும் பல்வேறு தலைப்புகள் பற்றிய ஒரு அடிப்படை விளக்கத்தை இப்புத்தகத்தில் தந்துள்ளேன். இதில் குறிப்பிட்டுள்ள ஒவ்வொன்றைப் பற்றியும் இன்னும் தேடித் தேடி ஆழமாகக் கற்று நிறைய பயிற்சிகள் செய்து

பார்த்து கற்றுத் தேற வேண்டியது வாசகர்களின் பொறுப்பு.

எளிய தமிழில் *Machine Learning* என்று ஏற்கெனவே ஒரு புத்தகத்தை எழுதி வெளியிட்டுள்ளேன். அதைப் படித்து முடித்த பின்னர் இப்புத்தகத்தை படிக்கத் தொடங்குவதே நல்லது. எடுத்தவுடன் இப்புத்தகத்திலிருந்து ஆரம்பிக்க வேண்டாம்.

http://kaniyam.com/learn-machine-learning-in-tamil என்ற முகவரியில் இருந்து அந்த நூலை பதிவிறக்கம் செய்யலாம்

து. நித்யா

கிழக்கு தாம்பரம்
17 *மார்ச்* 2020

மின்னஞ்சல்:
nithyadurai87@gmail.com

வலை பதிவு:
http://nithyashrinivasan.wordpress.com



1 Deep Learning

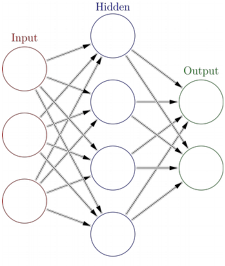
இயந்திர வழிக் கற்றலின் (Machine Learning) ஒரு பகுதியாக நியூரல் நெட்வொர்க்ஸ் என்பது அமையும். அதாவது மனிதனுடைய மூளை எவ்வாறு கற்கிறது என்பதை முன்னோடியாகக் கொண்டு உருவாக்கப்பட்டதே நியூரல் நெட்வொர்க்ஸ் ஆகும். முதலில் ஒரு குழந்தை பிறக்கும்போது அதனுடைய மூளைக்கு ஒன்றுமே தெரியாது. சுழியத்திலிருந்து ஆரம்பித்து பின்னர் ஒவ்வொரு விஷயமாகக் கற்கிறது. அதாவது குழந்தையின் மூளையிலுள்ள ஒரு மூளை நரம்பு(நியூரான்) ஒரு புதிய விஷயத்தைக் கற்கிறது. அடுத்ததாக மற்றொரு நரம்பு (மற்றொரு நியூரான்)ஏற்கெனவே கற்றுக் கொண்டுள்ள விஷயத்தோடு சேர்த்து இன்னொரு புதிய விஷயத்தையும் கற்றுக்

கொள்கிறது. இவ்வாறே நூறு பில்லியன் எண்ணிக்கையில் இருக்கும் பல்வேறு மூளைநரம்புகள் ஒன்றோடொன்று வலைப்பின்னல் வடிவில் பிணைக்கப்பட்டு (நியூரல் நெட்வொர்க்ஸ்) தொடர்ச்சியாக புதுப்புது விஷயங்களைக் கற்றுக் கொண்டே வருகின்றன. இதை அடிப்படையாக வைத்து உருவாக்கப்பட்டதே நியூரல் நெட்வொர்க்ஸ் எனும் தத்துவம் ஆகும்.

இத்தத்துவமானது பின்வரும் அமைப்பின் மூலம் செயற்கையாக நமது கணினியில் உருவாக்கப்படுகிறது. இவற்றில் முதல்நிலை, இடைநிலை, கடைநிலை எனும் 3 நிலைகளில் பல்வேறு நியூரான்கள் அமைக்கப்படுகின்றன. கணக்கீடுகளுக்காகப் பயன்படுத்தப்படும் முதன்மை அலகு நியூரான் என்று அழைக்கப்படும். ஒவ்வொரு நியூரானும் வெக்டர் எனும் அலகின் மூலம் தங்களுக்கான தரவுகளைப் பெற்றுக் கொள்கின்றன. முதல்

நிலையில் உள்ள நியூரான்கள் உள்ளீட்டுக்கானது. கடை நிலையில் உள்ள நியூரான்கள் வெளியீட்டுக்கானது. இடையில் ஒன்று அல்லது அதற்கு மேற்பட்ட மறைமுக அடுக்கில் பல்வேறு நியூரான்களை உருவாக்கி, ஒவ்வொரு அடுக்கிலும் தமக்குரிய அளவுருக்களை தரவுகளுடன் இணைத்து கணக்கீடுகளைத் தொடங்குகின்றன. அதாவது பயிற்ச்சிக்கான தரவுகளைப் பெற்று விளங்குவதே முதல்நிலை உள்ளீட்டு வெக்டர் ஆகும். இவ்வெக்டருடன் *weights* மற்றும் *bias*-ஐச் சேர்த்து இடைநிலையின் முதல் அடுக்கில் உள்ள நியூரான்கள் தனது கணக்கீடுகளைத் தொடங்குகின்றன. இம்மதிப்புகளுடன் மீண்டும் *weights* மற்றும் *bias*-ஐச் சேர்த்து இடைநிலையில் அடுத்த அடுக்கு காணப்படின், அது தனது கணக்கீடுகளைத் தொடங்கும். இவ்வாறே அடுத்தடுத்த நிலைகளில் கணக்கீடுகள் நிகழ்த்தப்பட்டு கடைசி நிலையில் உள்ள நியூரான் தனது *prediction*-ஐ வெளியிடுகிறது. இம்முறையிலேயே நியூரல் நெட்வொர்க்

தனக்கான புரிதலையும் கற்றலையும் மேற்கொள்கிறது.



Deep Learning என்பது இந்த நியூரல் நெட்வொர்கைப் பல்வேறு விதங்களில் அலசி ஆராய்கிறது. *Simple Neural Network, Shallow Neural Network, Deep Neural Network, Convolutional Neural Network, Recurrent Neural Network* போன்ற பல்வேறு விதங்களில் இவற்றை உருவாக்கி, தரவுகளை அலசி ஆராய்ந்து கற்றுக் கொள்கிறது. இவற்றை உருவாக்குவதற்கென *TensorFlow, PyTorch, Sonnet, Keras, Gluon, Swift, Chainer, DL4J, ONNX* போன்ற பல்வேறு கட்டமைப்புகள் *deep learning*-ல் உள்ளன. இவற்றில் ஒன்றான

TensorFlow வழியே *deep learning*-ல் உள்ள பல்வேறு அமைப்புகளை நாம் உருவாக்கிக் கற்கலாம். இதற்கென முதலில் *TensorFlow*-ல் உள்ள அடிப்படையான விஷயங்களைப் பற்றிக் கற்றுக்கொள்வோம்.

2 TensorFlow

Deep Neural Network-ன் பல்வேறு செயல்பாடுகளை ஆராய்வதற்கு உதவும் வகையில் *2011*-ல் *Google brain* குழுவால் உருவாக்கப்பட்ட ஒன்றே *Tensorflow* ஆகும். இது மிகவும் கடினமான கணித செயல்பாடுகளை சுலபமாக்குவதற்கு ஏற்ற வகையில் உருவாக்கப்பட்ட திறந்த மூல கணித *library* ஆகும். இதன் மூலம் தரவுகள் பற்றிய ஆழமான புரிதலை நாம் மேற்கொள்ள முடியும்.

தரவுகள் எந்த அமைப்பில் சேமித்து வைக்கப்படுகின்றதோ அந்த *data model*-ஆனது டென்சார் என்று அழைக்கப்படும் *(Data Model = Tensors)*. இதுபோன்ற பல்வேறு டென்சார்களுக்கிடையில் தரவுகள் பரிமாறப்பட்டு செயல்பாடுகள் நிகழ்வதையே *Tensorflow* என்று அழைக்கிறோம் *(Data flow =*

Tensor flow). இத்தகைய பரிமாற்றங்களைக் காண உதவும் கருவியே *Tensorboard* ஆகும் *(Dashboard = Tensor board).* இக்கருவியில் பல்வேறு டென்சார்களுக்கிடையில் தரவுகள் எவ்விதம் பரிமாறப்படுகின்றன என்பது ஒரு வரைபடம் மூலம் வரைந்து காட்டப்படுகிறது. இதுவே *Tensor flow graph* என்று அழைக்கப்படும். இவ்வரைபடம் *nodes* மற்றும் *edges* எனும் இரண்டு அம்சங்களைப் பெற்றிருக்கும். *Nodes* என்பது கணித செயல்பாடுகளையும் *edges* என்பது டென்சாரையும் குறிக்கும் *(Data flow graph = Tensor flow graph).* இது *CPU, GPU, Server, Mobile* போன்ற கருவிகளில் நிறுவுவதற்கு ஏற்ற வகையில் சுலபமான கட்டமைப்பினைப் பெற்றிருக்கும்.

உதாரண நிரல்:

"hello world" எனும் வாக்கியத்தை வெளிப்படுத்துவதற்கான உதாரணம் பின்வருமாறு அமையும். இதன் மூலம் ஒரு

மாதிரி tensor flow நிரலின் அமைப்பு முறையைத் தெரிந்து கொள்ளலாம். இது பொதுவாக 3 நிலைகளை உள்ளடக்கியது.

i) டென்சார்களின் உருவாக்கம் - Construction

ii) டென்சார்களை இயக்கி கணக்கீடுகளை நிகழ்த்துதல் - Running a session

iii) முடிவுகளை ஆராய்தல் - Routing & Analysis

https://gist.github.com/nithyadurai87/8c2d24e08cebd7dcff4696727c758913

```
import tensorflow as tf

a = tf.constant("hello world!")
print (a)

s = tf.Session()
print (s.run(a))
s.close()
```

```
with tf.Session() as s:
    print(s.run(a))
```

<u>நிரலுக்கான விளக்கம்:</u>

1. முதலில் *tensor flow library*-ஐ *tf* எனும் பெயரில் இறக்குமதி செய்துகொள்ள வேண்டும்.

2. பின்னர் இந்த *library*-க்குள் உள்ள அனைத்து *operators*-ஐயும் *tf.* எனக் கொடுத்து பயன்படுத்த வேண்டும். இங்கு *"hello world"* எனும் *constant*-ஐ வெளிப்படுத்த விரும்புகிறோம். எனவே *tf.constant()* என்பதற்குள் இச்சொற்றொடர் கொடுக்கப்பட்டு *a* எனும் டென்சார் உருவாக்கப்படுகிறது. *print (a)* எனக் கொடுக்கும்போது அது *constant* வகை டென்சாரை கொண்டுள்ளது என்பதை மட்டுமே வெளிப்படுத்தும். அதனுடைய மதிப்பினை வெளிப்படுத்தாது. இதுவே முதல் நிலையான டென்சார்களின் உருவாக்கம் ஆகும்.

3. அடுத்த நிலையில்தான் டென்சாரை இயக்கி மதிப்புகளை வெளிப்படுத்தப் போகிறோம். இதற்கு உதவுவதே *session()* எனும் *operator* ஆகும். இதை பின்வரும் இரண்டு விதங்களில் இயக்கலாம்.

விதம்1: *tf.session()* எனக் கொடுத்து *s* எனும் டென்சாரை உருவாக்கவும். பின்னர் *s.run()* எனக் கொடுத்து *constant*-ஐக் கொண்டுள்ள டென்சாரை இயக்கி மதிப்பினைக் காணலாம்.. *print (s.run(a))* என்பதே அந்த *constant* கொண்டுள்ள மதிப்பினை வெளிப்படுத்தும். கடைசியாக டென்சார்களை இயக்கி முடித்த பின்னர் நாம் உருவாக்கியுள்ள *session*-ஐ நிறுத்தி விட வேண்டும் *s.close()*. இது ஒரு விதம்.
விதம்2: *with tf.Session() as s:* எனக் கொடுத்து இதற்குள் நாம் இயக்க வேண்டிய அனைத்தையும் கொடுக்கலாம். இது மற்றொரு விதம். இம்முறையில் நாம் இயக்கும்போது கடைசியாக *session*-ஐ நிறுத்தத் தேவையில்லை.

நிரலுக்கான வெளியீடு:

Tensor("Const:0", shape=(), dtype=string)

b'hello world!'

b'hello world!'

.1 Constants

ஒரு குறிப்பிட்ட நிலையான மதிப்பினைப் பெற்று இயங்குவதற்கு *tf.constant()* எனும் *operator* பயன்படுகிறது. இது *string, int, float, bool* போன்ற பல்வேறு வகைகளில் தரவுகளைப் பெற்று இயங்கும் தன்மை உடையது. கீழ்க்கண்ட எடுத்துக்காட்டில் இதன் பல்வேறு தரவு வகைகளைக் காணலாம்

https://gist.github.com/nithyadurai87/d7ede47ace7ce9028342f8153b81c9b3

```
import tensorflow as tf

print (tf.constant("hello world!"))
print (tf.constant(1))
print (tf.constant(1, tf.int16))
print (tf.constant(1.25))
print (tf.constant([1,2,3]))
print (tf.constant([[1,2,3],
[4,5,6]]))
print (tf.constant([[[1,2,3],
[4,5,6]],
                    [[7,8,9],
[10,11,12]],
                    [[13,14,15],
[16,17,18]]]))
print (tf.constant([True, True,
False]))
print (tf.zeros(10))
print (tf.ones([10, 10]))
```

<u>நிரலுக்கான விளக்கம் & வெளியீடு:</u>

1. *“ hello world”* எனக் கொடுக்கும்போது, அது *string* வகை டென்சாரை உருவாக்கியுள்ளதைக் காணலாம். *dtype* என்பது *data type*-ஐக் குறிக்கும்.

Tensor("Const:0", shape=(), dtype=string)

2. எண் *1* எனக் கொடுக்கும்போது, *int32* வகை டென்சாரை உருவாக்கும். இதுவே *default* தரவு வகை ஆகும். இதையே *int16* என மாற்ற விரும்பினால், *1* எனக் கொடுக்கும்போதே பக்கத்தில், *int16* என அளிக்க வேண்டும்.

Tensor("Const_1:0", shape=(), dtype=int32)

Tensor("Const_2:0", shape=(), dtype=int16)

3. தசம எண்களைக் கொடுக்கும்போது *1.25*, *float* வகை டென்சாரை உருவாக்கும்.

Tensor("Const_3:0", shape=(), dtype=float32)

4. [1,2,3] எனும் *list* மதிப்பினைக் கொடுக்கும்போது, அதனை ஒரு பரிமாண *array-* ஆகக் கணக்கில் கொள்ளும். இதன் *shape* எனும் பண்பின் மதிப்பு காலியாக இல்லாமல், *3* என இருப்பதைக் காணலாம். அதாவது *1d array-*ல் *3 columns* உள்ளதை இது குறிக்கிறது.

Tensor("Const_4:0", shape=(3,), dtype=int32)

5. [[1,2,3],[4,5,6]] எனும் இரு பரிமாண *array-*ஐக் கொடுக்கும்போது, *shape* பண்பின் மதிப்பு *(2,3)* என உள்ளதைக் காணலாம். அதாவது கொடுக்கப்பட்ட *2d array-*ல் *2 rows* மற்றும் ஒவ்வொரு *row-*விலும் *3 columns* உள்ளது என்பதை இது குறிக்கிறது.

Tensor("Const_5:0", shape=(2, 3), dtype=int32)

6. [[[1,2,3],[4,5,6]] ,

[[7,8,9],[10,11,12]] ,

[[13,14,15],[16,17,18]]] எனும் முப்பரிமாண *array*-ஐக் கொடுக்கும்போது, *shape*-ன் மதிப்பு *(3,2,3)* என உள்ளதைக் காணலாம். முதலில் உள்ள *3* என்பது மூன்று *2d array*-ஐக் கொண்டுள்ளது என்பதைக் குறிக்கிறது. அடுத்து உள்ள *2,3* என்பது அத்தகைய *2d array*-ல் *2 rows, 3 columns* உள்ளது என்பதைக் குறிக்கிறது.

Tensor("Const_6:0", shape=(3, 2, 3), dtype=int32)

7. True, False என்பது போன்ற மதிப்புகளைக் கொடுக்கும்போது *bool* வகை டென்சாரை உருவாக்குவதைக் காணலாம்.

Tensor("Const_7:0", shape=(3,), dtype=bool)

8. tf.zeros மற்றும் *tf.ones* என்பது முறையே பூஜ்ஜியம் மட்டும் மற்றும் *1*-ஐ மட்டும் பெற்று விளங்கும் அணிகளை உருவாக்கப் பயன்படுகிறது. இதற்குள் கொடுக்கப்பட்டுள்ள மதிப்புகளைப் பொறுத்து இவை முறையே *10 columns*-ஐ மட்டும் பெற்று விளங்கும் *1d array*-ஐயும், *10 rows & 10 columns* -ஐப் பெற்று விளங்கும் *2d array*-ஐக் கொண்ட டென்சாரை உருவாக்கியுள்ளது.

Tensor("zeros:0", shape=(10,), dtype=float32)

Tensor("ones:0", shape=(10, 10), dtype=float32)

.2 Tensor properties

ஒரு டென்சார் என்பது அதனுடைய பெயர், வடிவம், தரவுவகை எனும் 3 பண்புகளைக் கொண்டு விளக்கப்படுகிறது.

பெயர் (Name): *ஒரு டென்சாரை உருவாக்கும்போதே, நமக்கு வேண்டிய பெயரைக் கொடுத்து உருவாக்கலாம். இல்லையெனில், எந்த operator-ஐப் பயன்படுத்தியுள்ளோமோ அதன் பெயரே டென்சாரின் பெயராக தானாக அமைந்துவிடும். அவ்வாறே ஒரே பெயரை மீண்டும் மீண்டும் பயன்படுத்தும்போதோ, அல்லது பெயரிடாதா ஒரே operator-ஐ மீண்டும் மீண்டும் பயன்படுத்தும்போதோ, அதன் பக்கத்தில் underscore (_) இட்டு முறையே 1,2,3, என வரிசை எண்கள் வெளிப்படுவதைக் காணலாம்.*

வடிவம் (Shape): *இப்பண்பு பொதுவாக shape=() காலி மதிப்பையே பெற்றிருக்கும். ஆனால் அணிகளைக் குறிப்பிடும்போது மட்டும், அது*

ஒரு பரிமாணமா, இருபரிமாணமா அல்லது முப்பரிமாணமா என்பதைக் குறிக்கப் பயன்படும். கீழ்க்கண்ட உதாரணத்தில் இப்பண்பினைப் பயன்படுத்தி, ஒரு டென்சாருடைய வடிவ அமைப்பைக் கொண்ட மற்றொரு டென்சார் உருவாக்கப்பட்டுள்ளது. அதாவது *d* எனும் டென்சாரில் உள்ளது போலவே *3 rows*-ஐக் கொண்ட *1*-ஐ மட்டும் பெற்று விளங்கும் *e* அணி இங்கு உருவாக்கப்பட்டுள்ளது. *tf.ones()* என்பது *1*-ஐ மட்டும் பெற்று விளங்கும் சதுர அணியை உருவாக்கும்.. எனவே *d.shape[0]* என *3 rows*-ஐ மட்டும் குறிக்கும் விதமாகக் கொடுத்தால் போதுமானது. அதற்கேற்ப *columns*-ஐ உருவாக்கி சதுர அணியை உருவாக்கிக் கொள்ளும்.

தரவுவகை (data type): *int, float, string, bool* போன்ற பல்வேறு தரவு வகைகளில் ஒரு டென்சார் எந்தத் தரவு வகையைச் சார்ந்தது என்பதைக் குறிக்க *dtype* என்பது பயன்படுகிறது. ஒரு தரவு வகையை மற்றொரு தரவு வகையாக மாற்றுவதற்கு *tf.cast()* எனும் *operator*

பயன்படுகிறது. கீழ்க்கண்ட உதாரணத்தில் *float* என்பது *int*-ஆக மாற்றப்பட்டுள்ளது.

https://gist.github.com/nithyadurai87/70cafef972154224c891220aec2336c3

```
import tensorflow as tf

a = tf.constant(1)
b = tf.constant(2)
c = tf.constant(1, name = "value")
print (a.name)
print (b.name)
print (c.name)

d = tf.constant([[1,2],[3,4],[5,6]])
e = tf.ones(d.shape[0])
print (a.get_shape())
```

```
print (d.get_shape())
print (e.get_shape())

c = tf.constant(1.25)
print (c.dtype)
print (tf.cast(c,
dtype=tf.int32).dtype)
```

நிரலுக்கான வெளியீடு:

Const:0

Const_1:0

value:0

()

(3, 2)

(3,)

<dtype: 'float32'>

<dtype: 'int32'>

.3 Operators

டென்சாரில் பல்வேறு ஆப்பரேட்டர்கள் உள்ளன. அவற்றில் சிலவான ஒர் எண்ணின் வர்க்க மூலம் காணுதல், இரு அணிகளின் கூட்டல், பெருக்கல் போன்றவற்றிற்குப் பயன்படும் ஆப்பரேட்டர்கள் கீழே கொடுக்கப்பட்டுள்ளன.

https://gist.github.com/nithyadurai87/811fb2f2d2871420f097c5c99c659f9b

```
import tensorflow as tf

a = tf.constant(121.5)
b = tf.constant([[1,2],[3,4],[5,6]])
c = tf.constant([[7,8],[9,10],
[11,12]])
```

```
r1 = tf.sqrt(a)
r2 = tf.add(b,c)
r3 = tf.multiply(b,c)

print (s.run(r1))
print (s.run(r2))
print (s.run(r3))
```

<u>நிரலுக்கான வெளியீடு:</u>

11.022704

[[8 10]

[12 14]

[16 18]]

[[7 16]

[27 40]

[55 72]]

.4 Variables

ஒரு variable என வரையறுக்கப்பட்ட டென்சாரின் மதிப்புகளை நம்மால் மாற்றி மாற்றி அமைக்க முடியும். நியூரல் நெட்வொர்க்கில் weights, bias போன்ற அளவுருக்களை நாம் மாற்றி மாற்றி அமைக்க வேண்டியிருக்கும். இது போன்ற இடங்களில் இந்த variable-ஐ நாம் பயன்படுத்திக் கொள்ளலாம்.

கீழ்க்கண்ட உதாரணத்தில், a எனும் variable உருவாக்கப்பட்டுள்ளது. இதனை உருவாக்கும்போது அதன் பெயர் (v1) மற்றும் [3,3] வடிவம் கொடுக்கப்பட்டுள்ளது. அதாவது 3 rows மற்றும் 3 columns-ஐக் கொண்ட ஒரு அணி உருவாக்கப்பட்டுள்ளது.. ஒரு session-ஐ உருவாக்கி அதில் a-ஐப் பிரிண்ட் செய்து பார்த்தால், random-ஆக இருக்கும் எண்களைப் பெற்ற அணி வெளிப்படுவதைக் காணலாம். மேலும், tf.global_variables_initializer() -ஐ இயக்குவதற்கு முன்னர் பிரிண்ட் செய்து

பார்த்தால், *"Attempting to use uninitialized value v1"* எனும் தவறு வெளிப்படுவதைக் காணலாம். ஆகவே இந்த *function*-ஐ இயக்கும்போதுதான் கொடுக்கப்பட்ட *variables*-அனைத்தும் அதன் துவக்க மதிப்பால் *initialize* செய்யப்படும். எந்த ஒரு *variable*-ஐயும் பயன்படுத்துவதற்கு முன்னர் இந்த *function*-ஐ இயக்குவது மிக மிக முக்கியம்.

அடுத்ததாக *b* எனும் *variable* உருவாக்கப்பட்டுள்ளது. இதனை உருவாக்கும்போதே *initializer* மூலம் அந்த *variable*-ல் என்ன மதிப்புகள் இருக்க வேண்டும் என நேரடியாகக் கொடுத்து விட்டோம். அதாவது என்ன வடிவத்தில் இருக்க வேண்டும் எனக் கூறுவதற்கு பதிலாக, அந்த அணியையே நேரடியாகக் கொடுத்து விட்டோம். இதனை பிரிண்ட் செய்து பார்த்தால், அந்த அணியின் மதிப்பு வெளிப்படுவதைக் காணலாம்.

கடைசியாக *x* எனும் *variable 1* எனும் துவக்க மதிப்பைப் பெற்று உருவாக்கப்பட்டுள்ளது. பின்னர் *session*-க்குள் *for loop* மூலம் *10* சுற்றுகளை

உருவாக்கி, ஒவ்வொரு சுற்றிலும் அந்த *variable, 2* எனும் *constant*-ஆல் பெருக்கப்படுகிறது. இவ்வாறு பெருக்கப்பட்டு கிடைத்த மதிப்பே, ஒவ்வொரு சுற்றின் இறுதியிலும், *x*-ன் மதிப்பாக *tf.assign()* மூலம் அமைக்கப்படுகிறது. எனவேதான் வெளியீடு *1*2=2, 2*2=4 , 4*2=8, 8*2=16 ...* என வந்துள்ளது.

https://gist.github.com/nithyadurai87/a2b16555bbe599d089969bf43318265e

```
import tensorflow as tf

a = tf.get_variable("v1", [3,3])
b = tf.get_variable("v2",
initializer=tf.constant([[3, 3],[4,
4]]))

x = tf.Variable(1)
y = tf.constant(2)
```

```
r = tf.assign(x,tf.multiply(x,y))

with tf.Session() as s:
    #print(s.run(a))

s.run(tf.global_variables_initialize
r())
    print(s.run(a))
    print(s.run(b))
    for i in range(10):
        print (s.run(r))
```

<u>நிரலுக்கான வெளியீடு:</u>

[[0.71750665 -0.39080024 -0.01946092]

[0.94270015 0.61512136 -0.7205548]

[0.6800127 -0.81497526 0.7290914]]

[[3 3]

[4 4]]

2

4

8

16

32

64

128

256

512

1024

.5 Placeholders

Placeholders என்பவை தரவுகள் வரவிருக்கின்றன எனும் குறிப்பை மட்டும் நமக்கு உணர்த்தப் பயன்படுகின்றன. உண்மையான தரவுகளை *session* இயங்கிக் கொண்டிருக்கும்போது *run-time* ல் பெற்றுக்கொள்கின்றன. *feed_dict* எனும் *argument* மூலமாக இவை தரவுகளைப் பெற்றுக்கொள்கின்றன. *Variables* என்பதற்கு ஏதாவதொரு துவக்க மதிப்பு தேவைப்படுகிறது. இதை வைத்துத் தான் பின்னர் இயங்கத் தொடங்கும். ஆனால் *placeholders* இயங்குவதற்கு எந்த ஒரு துவக்க மதிப்பும் தேவையில்லை. *session* இயங்கிக் கொண்டிருக்கும்போது மதிப்புகளை அளித்தால் போதுமானது. கீழ்க்கண்ட உதாரணத்தில், வெறும் பெயர் மற்றும் தரவுவகையைக் கொடுத்து *x, y* எனும் *2 placeholders* உருவாக்கப்பட்டுள்ளன. *x*-ன் மதிப்பை வைத்து *y*-ன் மதிப்பைக் கணக்கிடுவதற்கான விதியும் கொடுக்கப்பட்டுள்ளது. பின்னர் அதற்கான மதிப்புகள் *session* இயங்கிக்

கொண்டிருக்கும்போது x-க்கு 100, 200, 300 மற்றும் random முறையில் அமைந்த 1 முதல் 10 வரையிலான எண்கள் மாற்றி மாற்றி அளிக்கப்படுகின்றன. ஒவ்வொன்றுக்குமான y-ன் மதிப்பு கணக்கிடப்பட்டு அவை பிரிண்ட் செய்யப்பட்டுள்ளன.

[*https://gist.github.com/nithyadurai87/495e2484a602abc85d3e53ed7ace7407*](https://gist.github.com/nithyadurai87/495e2484a602abc85d3e53ed7ace7407)

```
import tensorflow as tf
import numpy as np

x =
tf.placeholder(tf.float32,name="x")
y = tf.placeholder(tf.float32,
[1],name="y")
z = tf.constant(2.0)
y = x * z

print (x)
with tf.Session() as s:
```

```
    print (s.run(y,feed_dict={x:
[100]}))
    print (s.run(y,{x:[200]}))
    print (s.run(y,{x:
np.random.rand(1, 10)}))
    print (s.run(tf.pow(x, 2),{x:
[300]}))
```

Tensor("x:0", dtype=float32)

[200.]

[400.]

[[1.5783005 0.30819204 0.26068646 1.8491662
1.0529723 1.7923148

0.9828862 1.5377688 0.06250755 1.2727137]]

[90000.]

பொதுவாக நியூரல் நெட்வொர்க்கில் பயிற்சியின் போது அளிக்கப்படும் மாதிரித் தரவுகளின் வடிவங்களை நம்மால் திட்டமிட்டுக் கூற முடியாது. இதுபோன்ற இடங்களில் *placeholders*-ஐப் பயன்படுத்தலாம். ஏனெனில் *variables*-ஐ வரையறுக்கும்போது அதன் வடிவத்தை நாம் திட்டமிட்டுக் கூற வேண்டியிருக்கும். அதாவது எத்தனை *rows & columns* இருக்கும் என்பதைக் கூறவேண்டி இருக்கும். இந்தப் பிரச்சனை *placeholders*-ல் இல்லை. *Run-time* ல் மாதிரித் தரவுகள் வர வர அதை அப்படியே ஒவ்வொரு நியூரானுக்கும் செலுத்துவதற்கு இவை பெரிதும் பயன்படுகின்றன.

.6 Tensor board

Tensor board என்பது பல்வேறு டென்சார்களுக்கிடையில் நிகழும் கணக்கீடுகளை வரைபடமாக வரைந்து காட்ட உதவும் கருவி ஆகும். கீழ்க்கண்ட எடுத்துக்காட்டில் *x1, y1, c* எனும் 3

டென்சார்களுக்கிடையில் நிகழ்ந்துள்ள கணக்கீடு பின்வருமாறு.

f = x1.y1 + squared(x1) + y1 + c

*= 5*6 + squared(5) + 6 + 5*

= 30 + 25 + 6 + 5

= 66

இவற்றை வரைபடமாக வரைந்து காட்ட *tf.summary.FileWriter()* எனும் *class* பயன்படுகிறது. இது *'tensorboard_example'* என்ற பெயரில் நம்முடைய தற்போதைய *directory*-ல் ஒரு *folder*-ஐ உருவாக்கும். இதற்குள் அனைத்து நிகழ்வுகளின் சுருக்கங்களையும் *(events & summaries)* சேமித்து வைக்கும். இதுவே *s.graph* மூலம் வரைபடமாக வரைந்து காட்டப்படும்.

[*https://gist.github.com/nithyadurai87/dbbe17d6036ea6188d9e581c3cbb2af6*](https://gist.github.com/nithyadurai87/dbbe17d6036ea6188d9e581c3cbb2af6)

```
import tensorflow as tf

x1 = tf.get_variable("a",
dtype=tf.int32,
initializer=tf.constant([5]))
y1 = tf.get_variable("b",
dtype=tf.int32,
initializer=tf.constant([6]))
c = tf.constant([5], name ="c")
f = tf.multiply(x1, y1) + tf.pow(x1,
2) + y1 + c

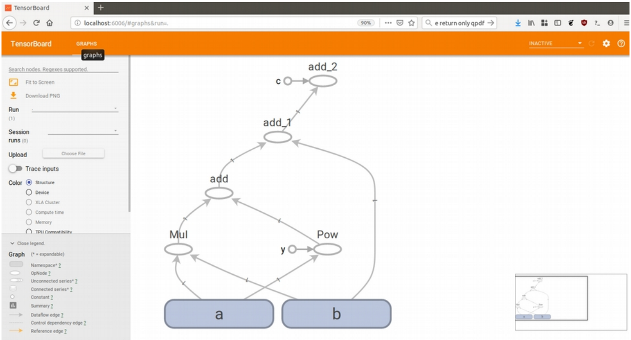
with tf.Session() as s:
    summary_writer =
tf.summary.FileWriter('tensorboard_e
xample',s.graph)

s.run(tf.global_variables_initialize
r())
    print (s.run(f))
```

இப்போது வரைபடத்துக்கான நிரலை எழுதி விட்டோம். அடுத்து *tensorboard*-ஐ இயக்க பின்வரும் கட்டளையை அளிக்கவும். இது *6006 port*-ல் இதனை இயக்கும். இதன் வெளியீடு *TensorBoard 1.13.1 at http://shrinivasan-Lenovo-Z50-70:6006 (Press CTRL+C to quit)* என்பது போன்று அமைந்தால், *tensorboard* இயங்கிக் கொண்டிருக்கிறது என்று அர்த்தம். அந்த *url*-ல் சென்று பார்த்தால் வரைபடம் காணப்படும்.

```
$ tensorboard --
logdir=tensorboard_example
```

கீழ்க்கண்ட வரைபடம் *Tensor flow graph* என்று அழைக்கப்படும். இது *nodes* மற்றும் *edges* எனும் இரண்டு அம்சங்களைப் பெற்றிருக்கும். *add, mul, pow* போன்றவை *nodes* என்று அழைக்கப்படும். இது கணித செயல்பாடுகளைக் குறிக்கும். *a,b,c* எனப் பெயர் கொண்ட *variables, constants* ஆகியவை *edges* என்று அழைக்கப்படும். இது டென்சாரைக் குறிக்கும்.


TensorBoard
localhost:6006/#graphs&run=.
TensorBoard
GRAPHS
graphs
INACTIVE
Search nodes. Regexes supported.
Fit to Screen
Download PNG
Run
Session runs
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
Close legend.
Graph
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
add_2
c
add_1
add
Mul
Pow
y
a
b

3 PyTorch

Deep Neural Network-ன் செயல்பாடுகளை ஆராய்வதற்கு உதவும் மற்றொரு வலிமையான கட்டமைப்பே *PyTorch* ஆகும். இது முகநூலின் செயற்கை அறிவுத்திறன் ஆய்வுக் குழு மூலம் உருவாக்கப்பட்ட பைதானை அடிப்படையாகக் கொண்ட ஒரு *library* ஆகும். *Torch* எனப்படும் இயந்திர வழிக்கற்றலுக்கான தொகுப்பின் அடிப்படையில் உருவானதே *pytorch* ஆகும்.

.1 Tensors

நியூரல் நெட்வொர்கைப் பொருத்தவரை தரவுகள் அனைத்தும் டென்சார் எனப்படும் கொள்கலனின் வழியேதான் செயல்படுகின்றன. இந்த பைடார்ச்சிலும் *torch.Tensor()* எனும் *class* மூலமாக தரவுகளை டென்சாராக

உருவாக்கலாம். கீழ்க்கண்ட உதாரணத்தில் *x1* எனும் பெயர் கொண்ட காலி டென்சார் உருவாக்கப்பட்டுள்ளது. பின்னர் *[1,2,3]* எனும் *1d array*-வைக் கொண்ட *x2* டென்சார் உருவாக்கப்பட்டுள்ளது. இது *3* உறுப்புகளைக் கொண்டுள்ளது என்பதனால், இதன் *size* மதிப்பு *3* என வெளிப்படுவதைக் காணலாம். மேலும் ஒவ்வொரு உறுப்பின் பக்கத்திலும், புள்ளியை இட்டு அனைத்தையும் *Float32* வகையில் உருவாக்கும். அவ்வாறு இல்லாமல் *int64* வகையில் உருவாக்க விரும்பினால், பெரிய *T*-க்கு பதிலாக சிறிய *t* -ஐ இட்டு *torch.tensor()* அல்லது *torch.as_tensor()* என்ற இரண்டில் ஏதாவது ஒன்றைப் பயன்படுத்தி உருவாக்கலாம். இவ்விரண்டையும் வைத்து உருவாக்கப்பட்ட *x3, x4* டென்சார்கள் ஒரே மாதிரி ஒரே வகையில் உருவாக்கப்பட்டுள்ளதைக் காணவும். எனவேதான் இவ்விரண்டு டென்சாரையும் கூட்டி அதன் மதிப்பு *([2, 4, 6])* வெளிப்படுத்தப்பட்டுள்ளது. அதுவே *x2*-ஐயும் *x3*-ஐயும் கூட்ட இயலாது.

ஒவ்வொரு டென்சாரும் *dtype, device, layout* எனும் *3* பண்புகளைப் பெற்றிருக்கும். *dtype* என்பது டென்சாரின் தரவு வகையை வெளிப்படுத்த உதவும். *device* என்பது ஒரு டென்சார் *CPU*-ல் இயங்கிக் கொண்டிருக்கிறதா அல்லது *GPU*-ல் இயங்கிக் கொண்டிருக்கிறதா என்பதை வெளிப்படுத்தும்.*layout* எனும் பண்பு ஒரு டென்சாருடைய நினைவக அமைப்பை *(memory layout)* வெளிப்படுத்தும். தற்போதைக்கு *torch.strided* எனும் மதிப்பையே இது வெளிப்படுத்தும். சோதனை முயற்சியாக *torch.sparse_coo* என்பது பயன்படுத்தப்பட்டு வருகிறது.

அடுத்ததாக *eye(), zeros(), ones()* போன்றவை முறையே முற்றொருமை அணி, பூஜ்ஜிய அணி, *1*-ஐ மட்டும் கொண்ட அணி ஆகியவற்றை கொடுக்கப்பட்டுள்ள வடிவத்தில் உருவாக்குகின்றன. பின்னர் *rand()* மூலம் *1d, 2d, 3d* போன்ற வடிவங்களில் *random*-ஆக அமைந்த எண்களைக் கொண்ட அணிகள் உருவாக்கப்பட்டுள்ளன.

https://gist.github.com/nithyadurai87/79d938591a3ddf6d2234dc54c441082c

```
import torch

x1 = torch.Tensor()
print (x1)

x2 = torch.Tensor([1,2,3])
print (x2)
print (x2.size())
print (x2.dtype)

x3 = torch.tensor([1,2,3])
x4 = torch.as_tensor([1,2,3])
print (x3)
print (x4)
print (x3+x4)
#print (x2+x3)

print (x4.dtype)
print (x4.device)
print (x4.layout)
```

```
print (torch.eye(2))
print (torch.zeros(2,2))
print (torch.ones(2,2))
print (torch.rand(2))
print (torch.rand(2,3))
print (torch.rand(2,3,4))
```

நிரலுக்கான வெளியீடு:

tensor([])

tensor([1., 2., 3.])

torch.Size([3])

torch.float32

tensor([1, 2, 3])

tensor([1, 2, 3])

tensor([2, 4, 6])

torch.int64

cpu

torch.strided

tensor([[1., 0.], [0., 1.]])

tensor([[0., 0.], [0., 0.]])

tensor([[1., 1.], [1., 1.]])

tensor([0.7487, 0.0652])

tensor([[0.6830, 0.9479, 0.7002],[0.3283, 0.8602, 0.6711]])

tensor([[[0.8002, 0.9752, 0.4617, 0.5603],[0.1520, 0.6906, 0.9570, 0.7589],[0.0122, 0.8932, 0.9644, 0.3375]],

[[0.5123, 0.3771, 0.5494, 0.1664],[0.0154, 0.4539, 0.8266, 0.8343],[0.8994, 0.5009, 0.0348, 0.0757]]])

.2 Some useful commands

topk() என்ற கட்டளைக்குள் k=1 என்றால் முதலாவது பெரிய மதிப்பையும், k=2 என்றால் முதல் இரண்டு பெரிய மதிப்புகளையும் வெளிப்படுத்தும். கீழ்க்கண்ட உதாரணத்தில் X எனும் 2d டென்சார் உருவாக்கப்பட்டுள்ளது. அதற்குள் dim=1 எனும்போது கொடுக்கப்பட்ட 3 columns-ஐ 0,1,2 என பரிமாணப்படுத்தி, முதல் row-ல் உள்ள 3 columns-ல் பெரிய மதிப்பான 120-ஐயும், 2 வது row-ல் உள்ள 3 columns-ல் பெரிய மதிப்பான 160-ஐயும் மற்றும் அவற்றின்

பரிமாணங்களையும் வெளிப்படுத்துகிறது. அதுவே *dim=0* எனும்போது கொடுக்கப்பட்ட *2 rows*-ஐ *0,1* என பரிமாணப்படுத்தி, முதல் *column*-ல் உள்ள *2 rows*-ல் பெரிய மதிப்பான *160*-ஐயும், *2* வது *column*-ல் உள்ள *2 rows*-ல் பெரிய மதிப்பான *120*-ஐயும், மூன்றாவது *column*-ல் உள்ள *2 rows*-ல் பெரிய மதிப்பான *90*-ஐயும் மற்றும் அவற்றின் பரிமாணங்களையும் வெளிப்படுத்துகிறது.

டென்சாரில் இருந்து ஒரு எண்ணை மட்டும் பிரித்து எடுக்க *torch.Tensor().item()* என்பது பயன்படுகிறது. *x.mul(2)* என்பது கொடுக்கப்பட்ட எண்ணை டென்சாரில் உள்ள அனைத்து மதிப்புகளுடனும் பெருக்கி வெளிப்படுத்தும். அதுவே எந்த ஒரு *operation*-ன் பக்கத்திலும் *underscore*-ஐ சேர்க்கும்போது *(mul_())*, அது வெளிப்படுத்தும் புதிய மதிப்புகளால் ஏற்கெனவே உள்ள மதிப்புகளை இடமாற்றம் செய்யும்.. இங்கு நாம் *underscore*-ஐப் பயன்படுத்தியுள்ளத்தால் இனிமேல் *x*-ன் மதிப்பு *2*-ஆல் பெருக்கிக் கிடைத்த புதிய

மதிப்புகளையே பெற்றிருக்கும். *reshape* செய்யும்போதும் இம்மதிப்புகளே மறுவடிவம் செய்யப்படுவதைக் காணலாம்.

ஒரு டென்சாரை *array*-ஆக மாற்ற அந்த டென்சாரின் பெயர் பக்கத்தில் புள்ளி வைத்து *numpy()* என்பது பயன்படுகிறது. அதுவே ஒரு *array*-வை டென்சாராக மாற்றுவதற்கு *torch.from_numpy()* என்பதற்குள் மாற்ற வேண்டிய *array()*-வைக் கொடுக்க வேண்டும்.

ஒரு டென்சாரை மறுவடிவம்*(reshape)* செய்வதற்கு *view()* கட்டளை பயன்படுகிறது. அதாவது கொடுக்கப்பட்ட *2d* டென்சாரை *1d*-ஆக மாற்றுவதற்கு, அந்த டென்சாரின் பக்கத்தில் புள்ளி வைத்து *view(1,6)* எனக் கொடுக்கலாம். அதாவது இரண்டு *rows*-ல் உள்ள மூன்று மூன்று *columns* அனைத்தும் ஒரே *row*-ல் வந்துவிடும். இல்லையெனில் *view(1,-1)* எனக் கொடுக்கலாம். *-1* எனும்போது, நாம் குறிப்பிட்டு இத்தனை உறுப்புகள் என்று கூறத் தேவையில்லை. அத்தனையும் மறுவடிவம் செய்யப்பட்டுவிடும்.

https://gist.github.com/nithyadurai87/8109fea71fd03258a494d4b8ee727a9a

```
import torch

x = torch.Tensor([[110,120,90],
[160,20,60]])

print (x.topk(k = 1, dim = 1))
print (x.topk(k = 2, dim = 1))
print (x.topk(k = 1, dim = 0))
print (x.topk(k = 2, dim = 0))

print (torch.Tensor([110]).item())
print (x.mul_(2))

y = x.numpy()
z = torch.from_numpy(y)
print (type(y))
print (type(z))

a = x.view(1,-1)
print(a)
```

```
print(a.size())
```

நிரலுக்கான வெளியீடு:

*torch.return_types.topk(values=tensor([[120.],
[160.]]),indices=tensor([[1],[0]]))*

*torch.return_types.topk(values=tensor([[120., 110.],
[160., 60.]]),indices=tensor([[1, 0],[0, 2]]))*

*torch.return_types.topk(values=tensor([[160., 120.,
90.]]),indices=tensor([[1, 0, 0]]))*

*torch.return_types.topk(values=tensor([[160., 120., 90.],
[110., 20., 60.]]),indices=tensor([[1, 0, 0],[0, 1, 1]]))*

110.0

```
tensor([[220., 240., 120.],[ 40., 320., 180.]])



<class 'numpy.ndarray'>

<class 'torch.Tensor'>

tensor([[220., 240., 120.,  40., 320., 180.]])

torch.Size([1, 6])
```

.3 GPU

CPU என்பது ஒரு கணினியின் மூளை என்றால், அந்த மூளையின் செயல்திறனை பலமடங்கு அதிகரிக்கும் வகையில் தற்போது உருவாக்கம் அடைந்திருப்பதே *GPU* ஆகும். இது *Graphics Processing Unit* எனப்படும். *Graphics, 3d games*

போன்றவற்றில் திரை ஒழுங்கமைவு செயல்களை அதிகளவு செய்வதற்கும், செயற்கை அறிவுத்திறன் துறைகளில் பல்வேறு கடின கணித செயல்பாடுகளை துரிதமாக செய்து முடிக்கவும் இந்த *GPU* பெரிதும் உதவுகிறது. இதன் துரித ஆற்றல்தான் இதனுடைய சிறப்பே!

கீழ்க்கண்ட உதாரணத்தில் *random*-ஆக தேர்ந்தெடுக்கப்பட்ட ஆயிரம் *rows & columns*-ஐக் கொண்ட *a* மற்றும் *b* எனும் இரண்டு *2d* அணிகள் உருவாக்கப்பட்டுள்ளன. பின்னர் இவை இரண்டும் *matmul()* மூலம் ஒன்றோடொன்று பெருக்கப்படுகின்றன. இந்த செயல் நடைபெறுவதற்கு முன்னர் உள்ள நேரமும் பின்னர் உள்ள நேரமும் முறையே *start* மற்றும் *end* எனும் இரு *variables* -ல் சேமிக்கப்படுகின்றன. பின்னர் இவை இரண்டுக்குமான வித்தியாசத்தைக் கண்டுபிடிப்பதன் மூலம் இந்த செயல் *CPU*-ல் நடைபெறுவதற்கான நேரம் கணக்கிடப்படுகிறது.

பின்னர் நமது கணினியில் *GPU* இருக்கிறதா என்பதை சோதிக்க *cuda.is_available()* எனும் கட்டளை பயன்படுகிறது. இது *True* என வெளிப்படுத்தினால், நமது *a, b* எனும் டென்சார் களை *GPU*-ல் செயல்படுவதற்கு ஏற்ற வகையில் மாற்ற *.cuda()* எனும் *function* உதவுகிறது. இதன் மூலம் அவை *GPU*-ல் ஏற்றப்பட்ட பின் மீண்டும் இவ்விரண்டு அணிகளின் பெருக்கலுக்கான நேரம் கணக்கிடப்படுகிறது. *CPU* -ல் இதற்கான நேரம் *22* விநாடிகள் என்றால், *GPU* -ல் இன்னும் துரிதமாக *0.0002* விநாடியில் இச்செயல் நடைபெற்று முடிந்திருப்பதைக் காணலாம்.

உங்களது கணினியில் *GPU* இல்லையெனில், *google colab* எனும் இணைப்பைப் (*https://colab.research.google.com/notebooks/welcome.ipynb*) பயன்படுத்தி கீழ்க்கண்ட சோதனையைச் செய்து பார்க்கலாம்.

https://gist.github.com/nithyadurai87/8eebce687ba439cd9b9691383c780dbe

```
import time
import torch

a = torch.rand(10000,10000)
b = torch.rand(10000,10000)
start = time.time()
a.matmul(b)
end = time.time()

print("{} seconds".format(end -
start))

print (torch.cuda.is_available())

a = a.cuda()
b = b.cuda()

start = time.time()
a.matmul(b)
end = time.time()
print("{} seconds".format(end –
start))
```

நிரலுக்கான வெளியீடு:

22.068870782852173 seconds

True

0.00020956993103027344 seconds

<class 'numpy.ndarray'>

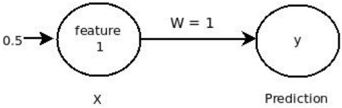
<class 'torch.Tensor'>

tensor([[220., 240., 120., 40., 320., 180.]])

torch.Size([1, 6])

4 Single Input Neuron

இப்பகுதியில் உள்ளீட்டு அடுக்கில் ஒரு நியூரானையும், வெளியீட்டு அடுக்கில் ஒரு நியூரானையும் வைத்து கணிப்பினை நிகழ்த்துவது எப்படி என்று பார்க்கலாம். இதனை நாம் *tensorflow* பயன்படுத்தி செய்து பார்க்கப் போகிறோம்.



[*https://gist.github.com/nithyadurai87/4ad051ef5c3cc2a3da6c043cd99d0f1b*](https://gist.github.com/nithyadurai87/4ad051ef5c3cc2a3da6c043cd99d0f1b)

```
import tensorflow as tf
```

```
X = tf.constant(0.5)
Y = tf.constant(0.0)
W = tf.Variable(1.0)

predict_Y = tf.multiply(X,W)

cost = tf.pow(Y - predict_Y,2)
min_cost =
tf.train.GradientDescentOptimizer(0.
025).minimize(cost)

for i in [X,W,Y,predict_Y,cost]:
    tf.summary.scalar(i.op.name,i)

summaries = tf.summary.merge_all()

with tf.Session() as s:
    summary_writer =
tf.summary.FileWriter('single_input_
neuron',s.graph)
```

```
s.run(tf.global_variables_initialize
r())
    for i in range(100):

summary_writer.add_summary(s.run(sum
maries),i)
        s.run(min_cost)
```

உள்ளீட்டு அடுக்கில் உள்ள நியூரான் உள்ளீட்டு மதிப்புடன் *weight* எனும் அளவுறுவினை இணைத்து தனது கணிப்பினை நிகழ்த்தும் என்று அறிவோம். இங்கு *0.5* எனும் உள்ளீட்டு மதிப்பினை எடுத்துக் கொண்டு, அதனுடன் *1.0* எனும் *weight* மதிப்பினை இணைத்து நிகழ்த்தியுள்ள கணிப்பு *predict_y* எனும் பெயரில் சேமிக்கப்பட்டுள்ளது. உண்மையான *0.0* எனும் வெளியீட்டு மதிப்பு *y* எனும் பெயரில் உள்ளது. இவ்விரண்டுக்கும் உள்ள வேறுபாடே *cost* என்று அழைக்கப்படுகிறது. இதனைக் கணக்கிட பழைய முறையான *square method* -ஐ இங்கு

பயன்படுத்தியுள்ளோம். *cost* மதிப்பினை குறைப்பதற்கான *gradient descent* எனும் தத்துவத்தை டென்சாரில் ஒற்றை வரியில் செயல்படுத்தி விடலாம். இது *learning_rate* எனும் மதிப்பினை தனது *parameter*-ஆக எடுத்துக்கொண்டு *100* சுற்றுகளில் *cost*-ஐக் குறைக்க முற்படுகிறது.

அடுத்ததாக இவற்றையெல்லாம் டென்சார் திரையில் வெளிப்படுத்துவதற்கான நிரல் கொடுக்கப்பட்டுள்ளது. எந்தெந்த மதிப்புகளையெல்லாம் திரையில் வெளிப்படுத்த விரும்புகிறோமோ, அவற்றையெல்லாம் ஒரு *list*-க்குள் அமைத்து *for loop*-மூலம் தொடர்ச்சியாக *scalar_summary()* எனும் *function*-க்குள் செலுத்துகிறோம். இதன் *arguments*-ஆக *list*-ல் உள்ள ஒவ்வொரு டென்சாரின் பெயரும், அதன் கீழ் அமைந்துள்ள மதிப்புகளும் கொடுக்கப்பட்டுள்ளன. அதாவது ஒவ்வொரு டென்சாரின் கீழும் அமைந்துள்ள மதிப்புகளின் சுருக்க நெறிமுறையே *(summary protocol buffer)* இதன் வெளியீடாக அமைகிறது.

இந்த சுருக்க நெறிமுறைகளின் அடிப்படையில் சுருக்கம் செய்யப்பட்ட அனைத்து டென்சார்களையும் ஒருங்கிணைப்பதற்கு *summary.merge_all()* பயன்படுகிறது.

பின்னர் ஒரு *session*-ஐ உருவாக்கி ஒருங்கிணைக்கப்பட்ட அனைத்து டென்சார்களின் சுருக்க மதிப்புகளையும் வரைபடமாக வரைந்து காட்ட *tf.summary.FileWriter()* பயன்பட்டுள்ளது. இது தற்போதைய *directory*-ல் *'single_input_neuron'* என்ற பெயரில் ஒரு *folder*-ஐ உருவாக்கும். இதற்குள் தான் அனைத்து டென்சார்களின் *summary* மதிப்புகளும் கோப்பு வடிவில் *asynchronous* முறையில் சேர்க்கப்படும். பின்னர் *for loop* மூலம் *100* சுற்றுகளை உருவாக்கி, ஒவ்வொரு சுற்றிலும் *0.025* எனும் கற்றல் விகிதத்தின் அடிப்படையில் அமைந்த புதிய *parameter* மதிப்புகளுக்கு *summaries*-ஐ உருவாக்கி இணைக்கிறது. டென்சார் திரையில் வரைபடத்தைக் காண்பதற்கான பதிவுக்

கோப்புகள் *(log files)* அனைத்தும் இம்முறையிலேயே உருவாக்கப்படும்.

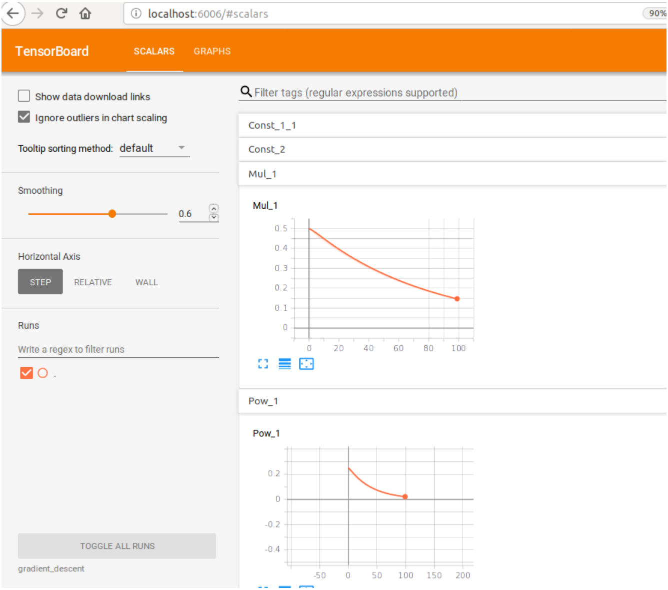
அடுத்ததாக கீழ்க்கண்ட கட்டளையை இயக்கி டென்சார் திரையில் சென்று பார்க்கவும்.

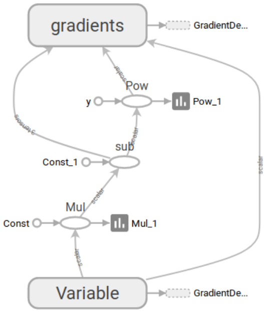
```
$ tensorboard --
logdir=single_input_neuron
```

TensorBoard 1.13.1 at http://shrinivasan-Lenovo-Z50-70:6006 (Press CTRL+C to quit)

திரையில் *Scalars* மற்றும் *Graphs* எனும் இரண்டு பிரிவுகள் காணப்படும். *Scalar* எனும் பிரிவில் *list*-க்குள் நாம் கொடுத்து வெளிப்படுத்தச் சொல்லிய ஒவ்வொரு மதிப்புகளுக்குமான வரைபடம் பின்வருவதுபோல் காணப்படும். எடுத்துக்காட்டாக *predict_y* தாங்கியுள்ள மதிப்பு *100* சுழற்சிகளில், ஒவ்வொரு சுழற்சியிலும் *0.0* எனும் உண்மையான *y*-மதிப்புக்கு நெருக்கத்தில் எவ்வாறு குறைந்து கொண்டே வருகிறது

என்பது வரைந்து காட்டப்பட்டுள்ளது. அடுத்ததாக *Graphs* எனும் பகுதியில் டென்சார்களுக்கிடையில் நடைபெற்ற கணக்கீடுகளுக்கான வரைபடம் வரைந்து காட்டப்பட்டுள்ளது.


localhost:6006/#scalars
90%
TensorBoard
SCALARS
GRAPHS
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
gradient_descent
Filter tags (regular expressions supported)
Const_1_1
Const_2
Mul_1
Mul_1
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
Pow_1
Pow_1
0.2
0
-0.2
-0.4
-50
0
50
100
150
200


gradients
GradientDe...
Pow
y
Pow_1
sub
Const_1
scalar
Mul
Const
Mul_1
Variable
GradientDe...


Variable
gradients
GradientDesc...

.1

5 Neural Networks

சென்ற எடுத்துக்காட்டில் உள்ளீட்டு அடுக்கில் உள்ள ஒரு நியூரானையும், வெளியீட்டு அடுக்கில் உள்ள ஒரு நியூரானையும் இணைத்து கணிப்பு எவ்வாறு நடக்கிறது என்று பார்த்தோம். இப்போது உள்ளீட்டு அடுக்கில் பல நியூரான்களை அமைத்து அவற்றை வெளியீட்டு அடுக்கில் உள்ள ஒரு நியூரானுடன் இணைத்து கணிப்பினை எவ்வாறு நிகழ்த்துவது என்று பார்க்கலாம். முதலில் இதன் தத்துவார்தங்களை சாதாரண பைதான் நிரல் கொண்டு எழுதிப் புரிந்து கொள்வோம். பின்னர் அதற்கு இணையான டென்சார் நிரலை எவ்வாறு பயன்படுத்துவது என்று பார்க்கலாம்.

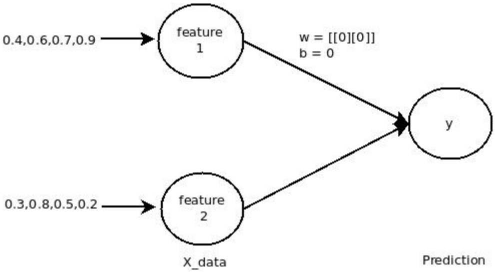
.1 Python code

நியூரல் நெட்வொர்கில் உள்ள பல்வேறு நியூரான்கள் மாதிரித் தரவுகளுடன் அவற்றுக்கே உரிய பல்வேறு அளவுருக்களை இணைத்து இணைத்து கணிப்புகளை நிகழ்த்துகிறது. முதலில் அளவுருக்ககளின் (parameters – weights, bias) மதிப்புகளை சுழியம் என வைத்து கணிப்புகளை நிகழ்த்திப் பார்க்கும். இந்நிலையில் குறைந்த அளவு கணிப்புகளே சரியாகவும், நிறைய கணிப்புகள் தவறாகவும் அமையும் பட்சத்தில் அளவுருக்களின் மதிப்புகளை மாற்றி மாற்றி சரியாகக் கணிக்க முயல்கிறது. கடைசியாக ஓரளவுக்கு கணிப்புகள் அனைத்தும் சரியாக அமைந்துவிடும் பட்சத்தில், தனது கற்றலை நிறுத்திக் கொள்கிறது. இத்தகைய கடைசி நிலையில் நாம் பயன்படுத்தியுள்ள அளவுருக்களின் மதிப்பையே எதிர்காலத்தில் வரப்போகின்ற தரவுகளை கணிப்பதற்கு நாம் பயன்படுத்திக் கொள்ளலாம்.

அனைத்து மாதிரித் தரவுகளின் உள்ளீடுகளை வைத்துக் கொண்டு அதன் வெளியீடுகளை சரியாகக் கணிப்பதற்கு, சுழியத்தில் ஆரம்பித்து, கடைசி நிலையை அடையும் வரை அளவுருக்களின் மதிப்பை மாற்றி மாற்றி கணிக்கும் முறையே "முன்னோக்கிப் பரவுதல்" / *Forward propagation* என்று அழைக்கப்படுகிறது. அளவுருக்களின் மதிப்பை ஒருசில காரணிகளின் அடிப்படையில் மாற்றும் முறைக்கு பின்னோக்கிப் பரவுதல் / *Back propagation* என்று பெயர். இவ்வாறாக முன்னோக்கிப் பின்னோக்கிப் பரவுதல் மூலம் நியூரல் நெட்வொர்க்கானது தனது கற்றலை நிகழ்த்துகிறது.

கீழ்க்கண்ட எடுத்துக்காட்டில் உள்ளீட்டுத் தரவு ஒரு *2d array*-வைக் கொண்டுள்ளது. அவற்றில் *4 rows* மற்றும் ஒவ்வொரு *row*-விலும் *2 columns* உள்ளன. அதாவது *2 features*-ஆல் விளக்கப்படும் *4* மாதிரித் தரவுகளைக் கொண்டுள்ளது. இந்த *2 features*-ம் உள்ளீட்டு அடுக்கில் அமையும் *2* நியூரான்களாக அமையும். இதற்கான வெளியீடு

0 மற்றும் *1*-ஆக உள்ளது. எனவே இது *logistic regression*-க்கான எடுத்துக்காட்டு என்பதால், இதற்கான வெளியீட்டு *layer*-ல் *sigmoid activation fn* பயன்படுத்தப்பட்டுள்ளது.



https://gist.github.com/nithyadurai87/75e91a68e07d2e375aaf3ba4c87ef49a

```
from sklearn import datasets
import numpy as np

X_data = np.array([[0.4,0.3],
[0.6,0.8],[0.7,0.5],[0.9,0.2]])
Y_data = np.array([[1],[1],[1],[0]])

X = X_data.T
Y = Y_data.T
```

```
W = np.zeros((X.shape[0], 1))
b = 0

num_samples = float(X.shape[1])
for i in range(1000):
    Z = np.dot(W.T,X) + b
    pred_y = 1/(1 + np.exp(-Z))
    if(i%100 == 0):
      print("cost after %d 
epoch:"%i)
      print (-1/num_samples 
*np.sum(Y*np.log(pred_y) + (1-
Y)*(np.log(1-pred_y))))
    dW = 
(np.dot(X,(pred_y-Y).T))/num_samples
    db = 
np.sum(pred_y-Y)/num_samples
    W = W - (0.1 * dW)
    b = b - (0.1 * db)

print (W,b)
```

நிரலுக்கான விளக்கம்:

மாதிரித் தரவுகள் (Sample data):

கீழ்க்கண்ட உதாரணத்தில் உள்ளீடாக 4 மாதிரித் தரவுகளும் (X_data), அதற்கான வெளியீடாக 4 மாதிரித் தரவுகளும் (Y_data) பயிற்சிக்கு அளிக்கப்பட்டுள்ளன. இதன் வெளியீடானது 0 & 1 என இருப்பதால், இது logistic regression-க்கானது என நாம் தெரிந்து கொள்ளலாம். உள்ளீட்டுக்கான தரவானது 4 rows & 2 columns கொண்ட 2d array ஆக உள்ளது. அதாவது 4 sample data-க்கான 2 features கொடுக்கப்பட்டுள்ளது.

X_data = np.array([[0.4,0.3],[0.6,0.8],[0.7,0.5],[0.9,0.2]])

Y_data = np.array([[1],[1],[1],[0]])

உள்ளீட்டு அடுக்கு (Input Layer):

நியூரல் நெட்வொர்க்-ஐப் பொருத்த வரையில் முதல் அடுக்கில் உள்ளீட்டுத் தரவுகளுக்கான *features* அமைந்திருக்கும். இந்த *features*-ன் எண்ணிக்கையில் அமைந்த *weights & bias* அணி உருவாக்கப்பட்டு கணக்கீடுகள் நிகழும். எனவே உள்ளீட்டுத் தரவுகளை *transpose* செய்து *features*sample_records* என்று அமையுமாறு மாற்றிக் கொள்ள வேண்டும் *(2,4)*. அவ்வாறே வெளியீட்டுத் தரவுகளைக் கொண்ட அணியின் *shape*-ம் *(4,)* என இருக்கும். இதையும் *transpose* செய்து *(1,4)* என மாற்றிக் கொள்ள வேண்டும். இவ்வாறே *X, Y* உருவாக்கப்படுகிறது.

X = X_data.T

Y = Y_data.T

அளவுருக்களின் துவக்கநிலை மதிப்புகள் *(Initializing weights & bias):*

இப்போது *X.shape[0]* என்பது *2 features*-ஐயும், *X.shape[1]* என்பது பயிற்சிக்கு அளிக்கப்பட்டுள்ள *4* மாதிரித் தரவுகளையும் குறிக்கிறது. *weights*-ஐ *0* என *initialize* செய்வதற்கு *np.zeros()* பயன்படுகிறது. இந்த அலகு மாதிரித் தரவுகளில் உள்ள *features*-ன் எண்ணிக்கைக்கு இணையாக இருக்க வேண்டும் என்பதற்காக *X.shape[0]* என கொடுக்கப்பட்டு பூஜ்ஜிய அணி உருவாக்கப்பட்டுள்ளது. *bias*-க்கு துவக்க மதிப்பாக பூஜ்ஜியம் அளிக்கப்படுகிறது.

W = np.zeros((X.shape[0], 1))

b = 0

சகாப்தங்கள் (Epochs):

X.shape[1] என்பது பயிற்சிக்கு அளிக்கப்பட்டுள்ள *4* மாதிரித் தரவுகளைக் குறிக்குமாதலால் இதை வைத்து *num_samples* எனும் *variable*

உருவாக்கப்படுகிறது. இது *cost* கண்டுபிடிக்கப் பயன்படுகிறது. அதாவது மொத்தத் தரவுகளில் எவ்வளவு தரவுகளுக்கு கணிப்புகள் தவறாக நிகழ்ந்துள்ளது என்பதைக் கணக்கிடப் பயன்படுகிறது. முதலில் பூஜ்ஜியம் என வரையறுக்கப்பட்ட அளவுருக்களை இணைத்து கணிப்புகளை நிகழ்த்தி *cost* கண்டுபிடிக்கிறது. இந்த *cost* அதிகமாக இருக்கும் பட்சத்தில் (அதாவது அதிக அளவு கணிப்புகள் தவறாக நிகழ்ந்திருந்தால்) கொடுக்கப்பட்ட அளவுருக்களை மாற்றி மீண்டும் கணித்து அதற்கான *cost* கண்டுபிடிக்கிறது. இவ்வாறே *for loop* மூலம் *1000* சுற்றுகள் செல்கின்றன. ஒவ்வொரு சுற்றும் *1 epoch* / சகாப்தம் என்றழைக்கப்படுகிறது. ஒவ்வொரு *epoch*-லும் முன்னோக்கிப் பரவி தரவுகள் அனைத்தையும் கணிக்கும் முறையும், கணிக்கப்பட்ட தரவுகளுக்கான இழப்பைக் கணக்கிடும் முறையும் , இழப்பு அதிகமாக இருக்கும் பட்சத்தில் பின்னோக்கிப் பரவுதல் முறைப்படி அளவுருக்களை மாற்றும் நிகழ்வும் தொடர்ச்சியாக நடைபெறுகின்றன.

முன்னோக்கிப் பரவுதல் (*Forward Propagation*):

ஒரு நியூரான் *input features*-ல் உள்ள மதிப்புகளுடன் *weights* மற்றும் *bias*-ஐச் சேர்த்து தனது கணக்கீடுகளைத் தொடங்கும் என ஏற்கெனவே பார்த்தோம். இங்கும் *np.dot()* மூலம் உள்ளீட்டுத் தரவுடன் சுழியம் எனும் துவக்க மதிப்பைப் பெற்ற *weights* மற்றும் *bias*-ஐ இணைத்து *Z* -ஐக் கணக்கிடுகிறது. இந்த *Z* என்பது ஒரு நியூரான் கணக்கிட்ட *linear* முறையில் அமைந்த கணிப்புகள் ஆகும். இதனை *logistic regression*-க்கு ஏற்ற முறையில் அமைக்க, *Z* மதிப்பானது *sigmoid activation function*-க்குள் செலுத்தப்பட்டு *0 / 1* என கணிக்கப்படுகிறது. இதே போன்று ஒவ்வொரு முறையும், மாற்றப்பட்ட அளவுருக்களை இணைத்து கணிப்புகளை நிகழ்த்தும் முறையே *forward propagation* என்று அழைக்கப்படுகிறது.

Z = np.dot(W.T,X) + b

pred_y = 1/(1 + np.exp(-Z))

இழப்பைக் கணக்கிடுதல் (Finding cost):

சுழியம் என அளவுருக்கள் (parameters = weights/bias) இருக்கும் போது, ஒரு நியூரான் கணிக்கின்ற விஷயம் தவறாக அமைவதற்கான வாய்ப்பு எந்த அளவுக்கு உள்ளது என்பதைக் கணக்கிட்டு வெளிப்படுத்துவதற்கான நிரல் பின்வருமாறு. இதே போன்று ஒவ்வொரு சுற்றிலும் அளவுருக்கள் மாற்றப்பட்டு, கணிப்புகள் நிகழ்த்தப்பட்டு இந்த cost கணக்கிடப்படுகிறது. மொத்தம் 1000 சுற்றுகள் என்பதால் ஒவ்வொரு முறையும் இம்மதிப்புகள் print செய்யப்படுவதைத் தவிர்க்க, சுற்றுகளின் எண்ணிக்கை 100-ன் மடங்காக இருக்கும்போது மட்டுமே இதனை print செய்க எனக்கூற if-ஐப் பயன்படுத்தியுள்ளோம். இதனால் மொத்தம் 10 முறை மட்டுமே cost வெளியிடப்படுகிறது.

```
if(i%100 == 0):

    print("cost after %d epoch:"%i)

    print (-1/num_samples *np.sum(Y*np.log(pred_y) + (1-Y)*(np.log(1-pred_y))))
```

பின்னோக்கிப் பரவுதல் (Backward Propagation):

ஏற்கெனவே உள்ள அளவுருக்களின் அடிப்படையில் இம்மதிப்பு பின்வரும் வாய்ப்பாடு மூலம் கணக்கிடப்படுகிறது. இம்முறையில் அளவுருக்களுக்கான *delta / derivative* எனும் மிகச் சிறிய மதிப்பு கணக்கிடப்படுகிறது. *derivative* என்றால் பெறுதி (மூலத்தினின்று பெறுகின்ற மதிப்பு) அல்லது வழித்தோன்றல் என்று பொருள் கொள்ளலாம். *Gradient descent* எனும் பகுதியில் இம்மதிப்பு எவ்வாறு கணக்கிடப்படுகிறது என்பதைக் கண்டோம். அதாவது உண்மையான

மதிப்புக்கும், கணிக்கப்பட்ட மதிப்புக்குமிடையே சாய்வான ஒரு கோடு வரையப்படுகிறது. இச்சாய்வுக் கோட்டின் மதிப்பே *derivative* என்றழைக்கப்படுகிறது. இதைக் கணக்கிடுவதற்கான வாய்ப்பாடு பின்வருமாறு.

dW = (np.dot(X,(pred_y-Y).T))/num_samples

db = np.sum(pred_y-Y)/num_samples

அளவுருக்களை மேம்படுத்துதல் *(updating parameters):*

நாம் கொடுத்துள்ள *learning rate*-ஆல் *back propagation* மூலம் நாம் கண்டறிந்த பெறுதி மதிப்புகளைப் பெருக்கி, ஏற்கெனவே உள்ள அளவுருக்களின் மதிப்பிலிருந்து கழித்து புதிய அளவுருக்களை நாம் உருவாக்கலாம். *learning rate* என்பது எந்த அளவுக்கு சிறியதாக நாம் அடுத்த

அடியை எடுத்து வைக்க வேண்டும் என்பதைக் குறிக்கும் (இங்கு 0.1 எனக் கொண்டுள்ளோம்).

*W = W - (0.1 * dW)*
*b = b - (0.1 * db)*

சரியான அளவுருக்களைக் கண்டுபிடித்தல்(Finding right parameters):

கடைசியாக கடைசி சுற்றில் பயன்படுத்தப்பட்ட அளவுருக்களின் மதிப்பு வெளிப்படுத்தப்படுகிறது. இதையே எதிர்காலத்தில் வரப்போகும் தரவுகளைக் கணிப்பதற்குப் பயன்படுத்திக் கொள்ளலாம்.

weights = [[-3.44] [4.40]]

bias = 1.66

நிரலுக்கான வெளியீடு:

cost after 0 epoch: 0.6931471805599453

cost after 100 epoch: 0.5020586179991661

cost after 200 epoch: 0.4448439151612328

cost after 300 epoch: 0.3979115275585397

cost after 400 epoch: 0.3590456846967788

cost after 500 epoch: 0.32654805177827606

cost after 600 epoch: 0.2990946818904699

cost after 700 epoch: 0.2756668906115973

cost after 800 epoch: 0.25548229634006936

cost after 900 epoch: 0.2379373586492477

[[-3.43906374] [4.4016814]] 1.6577403314904984

.2 TensorFlow code

மேற்கண்ட அதே விஷயத்தை *TensorFlow* மூலம் வைத்து எழுதியுள்ள நிரல் பின்வருமாறு. இதுவும் மேற்கண்ட அதே வெளியீட்டை வெளியிடும். இதில் பயன்படுத்தியுள்ள *functions*-ன் தெளிவான விளக்கங்களை *shallow neural network* எனும் பகுதியில் காணலாம்.

https://gist.github.com/nithyadurai87/fc1719d485dddfc4988c9e36975913c3

```
import tensorflow as tf

X_data = tf.constant([[0.4,0.3],
[0.6,0.8],[0.7,0.5],
[0.9,0.2]],name="input_value")
Y_data = tf.constant([[1.0],[1.0],
[1.0],[0.0]],name="output_value")
```

```
X = tf.transpose(X_data)
Y = tf.transpose(Y_data)

W =
tf.Variable(initial_value=tf.zeros([
1,X.shape[0]],
dtype=tf.float32),name="weight")
b =
tf.Variable(initial_value=tf.zeros([
1,1], dtype=tf.float32))

Z = tf.matmul(W,X) + b
cost =
tf.reduce_mean(tf.nn.sigmoid_cross_e
ntropy_with_logits(logits=Z,labels=Y
))
GD =
tf.train.GradientDescentOptimizer(0.
1).minimize(cost)

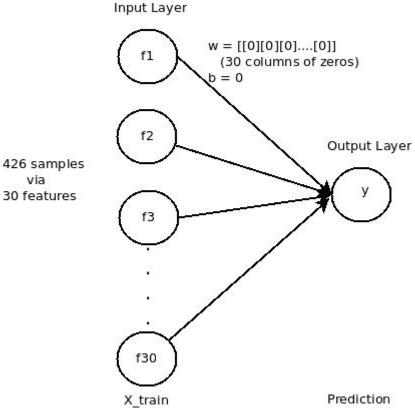
with tf.Session() as sess:
```

```
sess.run(tf.global_variables_initial
izer())
    for i in range(1000):
        c = sess.run([GD, cost])[1]
        if i % 100 == 0:
            print ("cost after %d
epoch:"%i)
            print(c)
    print (sess.run(W),sess.run(b))
```

6 Simple Neural Networks

இதுவரை நாம் பார்த்த அனைத்தும் புரிந்து கொள்ள சுலபமாக இருக்க வேண்டும் என்பதற்காக,, 1 நியூரான், 2 நியூரான் என்று சிறிய எண்ணிக்கையில் எடுத்துச் செய்து பார்த்தோம். இப்போது உண்மையாகவே 30 features-ல் அமையும் 426 பயிற்சித் தரவுகளை எடுத்து ஒரு நியூரல் நெட்வொர்க்கை உருவாக்கிப் பார்க்கப் போகிறோம். இதில் வெறும் உள்ளீடு மற்றும் வெளியீட்டுக்கான layer-ஐ மட்டுமே கொண்டிருக்கும். இடையில் எந்தஒரு hidden layer-ம் காணப்படாது.

கீழ்க்கண்ட எடுத்துக்காட்டில் *sklearn*-க்குள் உள்ள *datasets* என்பதற்குள் ஒருவருக்கு மார்பகப் புற்றுநோய் இருக்கா இல்லையா என்பதை முடிவு செய்வதற்கான மாதிரித் தரவுகளின் தொகுப்புகள் உள்ளன. இவை *426 rows* மற்றும் *30 features*-ஐக் கொண்டது. இவை *train_test_split* மூலம் பிரிக்கப்பட்டு *normalize* செய்யப்படுகின்றன. அவற்றில் ஒரு பாதியைக் கொடுத்து *4000* சுற்றுகளை உருவாக்கி, கற்றலுக்கான விகிதத்தை *0.75* என வைத்து நியூரல் நெட்வொர்க்குப் பயிற்சி அளித்து சரியான அளவுருக்களைக் கண்டுபிடிக்கிறோம். பின்னர் கண்டுபிடித்த அளவுக்களை வைத்து பயிற்சிக்கு அளித்த தரவுகளையும்*(X_train)*, அளிக்காத தரவுகளையும்*(X_test)* கணிக்கச் சொல்லுகிறோம். பயிற்சிக்கு அளித்த தரவுகளின் துல்லியத்தன்மை *98%* எனவும், பயிற்சிக்குச் செலுத்தாத மீதித் தரவுகளை எதிர்காலத் தரவுகலாகக் கருதி அதனைக் கணித்து வரும் துல்லியத்தன்மை *93%* எனவும் வெளிப்படுவதைக் காணலாம்.



https://gist.github.com/nithyadurai87/6c39697e3134a7ba5bd5cbe8790f95db

```
from sklearn import datasets
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

```
def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min,
col_max - col_min)

def model(X,Y):
  num_samples = float(X.shape[1])
  W = np.zeros((X.shape[0], 1))
  b = 0
  for i in range(4000):
    Z = np.dot(W.T,X) + b
    A = 1/(1 + np.exp(-Z))
    if(i%100 == 0):
      print("cost after %d
epoch:"%i)
      print (-1/num_samples
*np.sum(Y*np.log(A) + (1-
Y)*(np.log(1-A))))
    dW =
(np.dot(X,(A-Y).T))/num_samples
    db = np.sum(A-Y)/num_samples
```

```
    W = W - (0.75 * dW)
    b = b - (0.75 * db)
  return W, b

def predict(X,W,b):
    Z = np.dot(W.T,X) + b
    i = []
    for y in 1/(1 + np.exp(-Z[0])):
        if y > 0.5 :
            i.append(1)
        else:
            i.append(0)
    return
(np.array(i).reshape(1,len(Z[0])))

(X_cancer, y_cancer) =
load_breast_cancer(return_X_y =
True)
X_train, X_test, y_train, y_test =
train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = normalize(X_train).T
```

```
y_train = y_train.T

X_test = normalize(X_test).T
y_test = y_test.T

Weights, bias = model(X_train,
y_train)

y_predictions =
predict(X_train,Weights,bias)
accuracy = 100 -
np.mean(np.abs(y_predictions -
y_train)) * 100
print ("Accuracy for training data:
{} %".format(accuracy))

y_predictions =
predict(X_test,Weights,bias)
accuracy = 100 -
np.mean(np.abs(y_predictions -
y_test)) * 100
```

```
print ("Accuracy for test data: {}
%".format(accuracy))
```

நிரலுக்கான வெளியீடு:

cost after 0 epoch:

0.6931471805599453

cost after 100 epoch:

0.24382767353051085

cost after 200 epoch:

0.1841491919513481 8

……………………..

cost after 3800 epoch:

0.060440634653931 39

cost after 3900 epoch:

0.05993526502299061

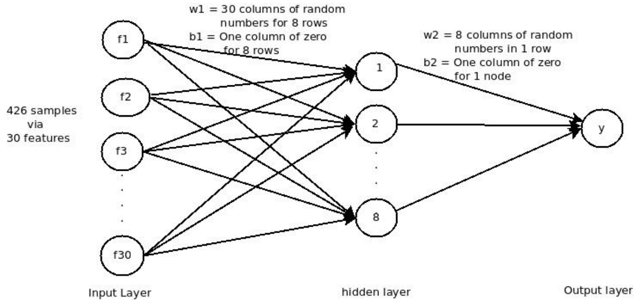
Accuracy for training data: 98.59154929577464 %

Accuracy for test data: 93.00699300699301 %

7 Shallow Neural Networks

Shallow என்றால் ஆழமற்ற என்று பொருள். *deep* என்றால் ஆழமான என்று பொருள். எனவே *Deep* நியூரல் நெட்வொர்கைப் பற்றிக் கற்பதற்கு முன்னர் இந்த *shallow* நியூரல் நெட்வொர்கைப் பற்றித் தெரிந்து கொள்வோம். இதற்கு முன்னர் நாம் பயன்படுத்திய மார்பகப் புற்றுநோய்க்கான உதாரணத்தையே இங்கும் பயன்படுத்திக் கொள்வோம். ஆனால் இதன் உள்ளீடு மற்றும் வெளியீட்டு *layer*-க்கிடையில் *hidden layer* ஒன்று காணப்படும். அதில் நாம் வரையறுக்கும் எண்ணிக்கையில் அமைந்த *nodes*-ஐப் பெற்றிருக்கும். இதன் வழியே செயல்படும்போது அதிக அளவிலான *features* குறைந்த எண்ணிக்கையில் மாற்றப்பட்டு கணிப்புகள் நிகழும்.

கீழ்க்கண்ட நிரலில் மாதிரித் தரவுகள் *train_test_split* மூலம் பிரிக்கப்பட்டு *normalize* செய்யப்படுவதற்கு முன்னர் , வரைபடம் ஒன்று வரைந்து பார்க்கப்படுகிறது. அதில் முதல் *2 features*-ஐ எடுத்துக் கொண்டு *(X_cancer[:,0], X_cancer[:,1])* அவற்றில் உள்ள தரவுகள் *0* அல்லது *1* எனும் வகைப்பாட்டில் *(c=y_cancer)* எவ்வகையின் கீழ் அமைகின்றன என்பது முறையே கரும்புள்ளிகளாகவும், மஞ்சள் புள்ளிகளாகவும் வரைந்து காட்டப்படுகிறது. இவை வரைபடத்தில் நேர்கோடு முறையில் பிரிக்க இயலாத தரவுகளாகக் கலந்திருப்பதைக் காணலாம். எனவே எளிய *logistic regression*-ஐப் பயன்படுத்துவது இதற்கு உதவாது. *shallow neural network* கொண்டு இதனை நாம் பிரிக்கப் போகிறோம். மேலும் இவற்றை *tensor flow* பயன்படுத்திச் செய்யப் போகிறோம். சென்ற எடுத்துக்காட்டில் நாம் அனைத்தையும் கோட்பாடுகளாக நிரல் எழுதி செய்தோம். இப்போது அவற்றுக்கான *tensor flow* செயல்பாட்டுத் திட்டங்களைப் பயன்படுத்தப் போகிறோம்.



https://gist.github.com/nithyadurai87/88a6f4e571200ce75de17464f9f13941

```
from sklearn.model_selection import
train_test_split
from sklearn.datasets import
load_breast_cancer
import tensorflow as tf
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

```
from matplotlib.colors import
ListedColormap

def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min,
col_max - col_min)

(X_cancer, y_cancer) =
load_breast_cancer(return_X_y =
True)

cmap =
matplotlib.colors.ListedColormap(['b
lack','yellow'])
plt.figure()
plt.title('Non-linearly separable
classes')
plt.scatter(X_cancer[:,0],
X_cancer[:,1], c=y_cancer, marker=
'o', s=50, cmap=cmap, alpha = 0.5 )
plt.show()
```

```
plt.savefig('fig1.png',
bbox_inches='tight')

X_train, X_test, Y_train, Y_test =
train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = normalize(X_train).T
Y_train = Y_train.reshape(1,
len(Y_train))

X_test = normalize(X_test).T
Y_test = Y_test.reshape(1,
len(Y_test))

num_features = X_train.shape[0]
X = tf.placeholder(dtype =
tf.float64, shape =
([num_features,None]))
Y = tf.placeholder(dtype =
tf.float64, shape = ([1,None]))
```

```
W1 =
tf.Variable(initial_value=tf.random_
normal([8,num_features], dtype =
tf.float64) * 0.01)
b1 =
tf.Variable(initial_value=tf.zeros([
8,1], dtype=tf.float64))
W2 =
tf.Variable(initial_value=tf.random_
normal([1,8], dtype=tf.float64) *
0.01)
b2 =
tf.Variable(initial_value=tf.zeros([
1,1], dtype=tf.float64))

Z1 = tf.matmul(W1,X) + b1
A1 = tf.nn.relu(Z1)
Z2 = tf.matmul(W2,A1) + b2

cost =
tf.reduce_mean(tf.nn.sigmoid_cross_e
ntropy_with_logits(logits=Z2,labels=
Y))
```

```
train_net =
tf.train.GradientDescentOptimizer(0.
2).minimize(cost)

init =
tf.global_variables_initializer()
with tf.Session() as sess:
  sess.run(init)
  for i in range(5000):
    c = sess.run([train_net, cost],
feed_dict={X: X_train, Y: Y_train})
[1]
    if i % 1000 == 0:
      print ("cost after %d
epoch:"%i)
      print(c)
  correct_prediction =
tf.equal(tf.round(tf.sigmoid(Z2)),
Y)
  accuracy =
tf.reduce_mean(tf.cast(correct_predi
ction, "float"))
```

```
  print("Accuracy for training data:", accuracy.eval({X: X_train, Y: Y_train}))
  print("Accuracy for test data:", accuracy.eval({X: X_test, Y: Y_test}))
```

<u>நிரலுக்கான விளக்கம்:</u>

Input Layer-ல் placeholders-ஐ உருவாக்குதல்:

பயிற்சிக்கு அளித்துள்ள தரவுகள் training, testing என்று பிரிக்கப்பட்டு, normalize மற்றும் reshape செய்யப்படுகின்றன. பின்னர், உள்ளீட்டு மற்றும் வெளியீட்டுத் தரவுகளை நியூரல் நெட்வொர்க்குச் செலுத்தி பயிற்சி அளிப்பதற்கான X, Y எனும் 2 placeholders உருவாக்கப்படுகின்றன. இவற்றின் வழியாகத்தான் தரவுகள் அனைத்தும் நியூரல் நெட்வொர்க்குச் செல்கின்றன. இவற்றில் அமையும் columns-ன் எண்ணிக்கை மாதிரித்

தரவுகளில் உள்ள *features*-ன் எண்ணிக்கையில் இருக்க வேண்டும் என்பதற்காக *X_train.shape[0]* என்பதும், *rows*-ன் எண்ணிக்கையில் நாம் குறிப்பாக எவ்வளவு தரவு மாதிரிகளை செலுத்தப் போகிறோம் என்பது தெரியாத பட்சத்தில் *None* என்பதும் *X*-ன் வடிவமைப்பில் கொடுக்கப்பட்டுள்ளன. அவ்வாறே *Y*-ன் வடிவமைப்பில், ஒரே ஒரு *target column*-ல் அமையும் பல்வேறு *records* என்பதைக் குறிக்கும் வகையில் *[1,None]* என்பது கொடுக்கப்பட்டுள்ளது.

X = tf.placeholder(dtype = tf.float64, shape = ([num_features,None]))

Y = tf.placeholder(dtype = tf.float64, shape = ([1,None]))

இப்போது நியூரல் நெட்வொர்க்கின் முதல் அடுக்கில் *X_train*-ல் உள்ள *30 features*-க்கான *placeholder nodes* அமைந்திருக்கும்.

Hidden Layer-க்கான அளவுருக்களை அமைத்தல்:

placeholders வழியேதான் ஒவ்வொரு *feature*-ல் உள்ள தரவுகளும் அடுத்த *layer*-க்குச் செலுத்தப்படும். அடுத்த *layer* தான் *hidden layer*. இது *shallow neural network* என்பதால் ஒரே ஒரு *hidden layer*-தான் இருக்கும். இவற்றில் அமையும் *nodes*-ன் எண்ணிக்கையை நாம் 8 என வரையறுத்துள்ளோம். எனவே இந்த 8 *nodes*-க்கான அளவுருக்கள் (*weights & bias*) *W1*, *b1* எனும் பெயரில் அமைந்துள்ளன.

W1 எனும் *variable*-க்கு *tf.random_normal()* மூலம் 30 *features*-க்கான அளவுருக்கள் ஒவ்வொரு *node*-க்கும் தனித்தனியாக *random* முறையில் தேர்ந்தெடுக்கப்பட்டு துவக்க மதிப்பாக அமைகிறது. அதாவது *hidden layer*-ல் உள்ள ஒவ்வொரு *node*-ம் *random*-ஆக தேர்ந்தெடுக்கப்பட்ட 30 *weights parameters*-ஐத் தங்கியிருக்கும். எனவே 30 *columns*-க்கான 8 *rows*-ஐக் கொண்ட ஒரு டென்சார் இதில் உருவாக்கப்படும்.

b1 எனும் *variable*-க்கு துவக்க மதிப்பாக பூஜ்ஜியம் என்பது அமையும். *hidden layer*-ல் உள்ள ஒவ்வொரு *node*-க்கும் இதனை அமைக்க *tf.zeros()* மூலம் ஒரே *column*-ல் *8* பூஜ்ஜியங்களைக் கொண்ட ஒரு டென்சார் உருவாக்கப்படுகிறது.

```
W1 =
tf.Variable(initial_value=tf.random_normal([8,num_featu
res], dtype = tf.float64) * 0.01)

b1 = tf.Variable(initial_value=tf.zeros([8,1],
dtype=tf.float64))
```

Output Layer-க்கான அளவுருக்களை அமைத்தல்:

கடைசியாக உள்ளது வெளியீட்டு *layer*. இதற்கான அளவுருக்கள் *W2, b2* எனும் பெயரில் அமைக்கப்படுகின்றன. வெளியீட்டு *layer* என்பது ஒரே ஒரு *node*-ஐக் கொண்டது. இதற்கு முந்தைய *hidden layer*-ல் உள்ள *8 nodes*

கணக்கிட்ட மதிப்புகளே இதற்கான *features.* ஆகவே இந்த *8 features*-க்கான ஒரு *node*-வுடைய அளவுருக்கள் *tf.random_normal([1,8])* மூலம் வரையறுக்கப்படுகின்றன. அவ்வாறே அந்த ஒரு *node*-வுடைய *bias* மதிப்பும் பூஜ்ஜியம் *tf.zeros([1,1]* என வரையறுக்கப்படுகிறது. *W2 = tf.Variable(initial_value=tf.random_normal([1,8], dtype=tf.float64) * 0.01)*

b2 = tf.Variable(initial_value=tf.zeros([1,1], dtype=tf.float64))

முன்னோக்கிப் பரவுதல் (Forward Propagation):

hidden layer-ல் உள்ள *8* நியூரான்கள் *placeholders*-வழியே வருகின்ற தரவுகளுடன் *weights* மற்றும் *bias*-ஐச் சேர்த்து தனது கணக்கீடுகளைத் தொடங்கும். இவ்வாறு கணக்கிட்டுக் கிடைத்த மதிப்பே *Z1* என்று அழைக்கப்படுகிறது. இவற்றை *0* அல்லது *1* எனும் வகையின் கீழ்

கணிக்க *sigmoid*-க்கு பதிலாக *relu* அல்லது *tanh* போன்ற ஏதாவதொன்றைப் பயன்படுத்தலாம். *hidden layer*-ல் உள்ள *8* நியூரான்களும் இதுபோன்ற *8* வகையான கணிப்புகளை வெளியிடுகின்றன. இதுவே *A1* ஆகும். இம்மதிப்புடன் *output layer*-ல் உள்ள *1* நியூரானானது அதனுடைய *weights* மற்றும் *bias*-ஐச் சேர்த்து *Z2*-ஐக் கணக்கிடுகிறது. இம்மதிப்பை மீண்டும் *0/1* -ன் கீழ் வகைப்படுத்தத் தேவையில்லை. *Tensor flow*-ஐப் பொருத்தவரை கடைசி *layer*-லிருந்து வெளிவரும் *activate* செய்யப்படாத மதிப்பையே இழப்பைக் கணக்கிடுவதற்கான *cost function*-க்குள் செலுத்த வேண்டும். பின்னர் அந்த *cost function* தனக்குள்ளாகவே *sigmoid* மூலம் *0/1* என *activate* செய்துவிடும்.

```
Z1 = tf.matmul(W1,X) + b1

A1 = tf.nn.relu(Z1)

Z2 = tf.matmul(W2,A1) + b2
```

இழப்பைக் கணக்கிடுதல் (Finding cost):

sigmoid_cross_entropy() என்பது *Tensor flow*-ல் இழப்பைக் கண்டுபிடிக்கப் பயன்படுகிறது. இதற்குள் கடைசி *layer*-லிருந்து வெளிவருகின்ற *activate* செய்யப்படாத *Z2* கணிப்புகளும், உண்மையான *Y* மதிப்புகளும் செலுத்தப்படுகின்றன. இவற்றை வைத்துக் கண்டுபிடித்த இழப்பின் சராசரியைக் கணக்கிட *tf.reduce_mean()* பயன்படுகிறது.

```
cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=Z2,labels=Y))
```

Gradient descent மூலம் இழப்பைக் குறைத்தல்:

Tensor flow-ல் *derivative*-ஐக் கணக்கிட்டு, அதனடிப்படையில் அளவுருக்களை மாற்றி, இழப்பினைக் குறைப்பதை ஒரே வரியில் செய்து விடலாம். *GradientDescentOptimizer()* எனும் *function* இதற்கான வேலையைச் செய்கிறது. *0.2* என்பது இதற்கு அளிக்கப்பட்டுள்ள *learning rate* ஆகும். சென்ற படியில் கண்டுபிடித்த *cost* மதிப்பு இதன் மீது செயல்படும் *minimize()* எனும் *function*-க்குள் செலுத்தப்படுகிறது.

train_net = tf.train.GradientDescentOptimizer(0.2).minimize(cost)

Session-ஐ உருவாக்கி அனைத்தையும் இயக்குதல்:

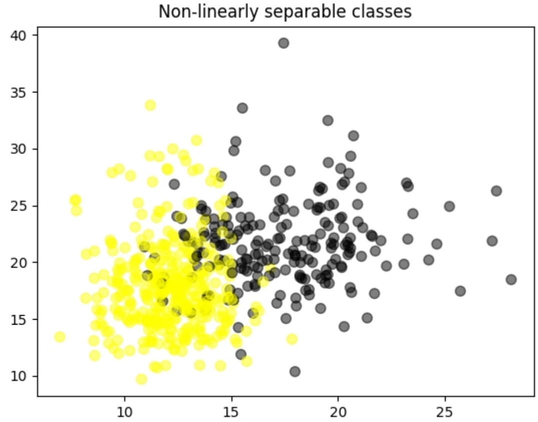
tf.global_variables_initializer() என்பது நம்முடைய நிரலில் நாம் *variable* என வரையறுத்துள்ள அனைத்தையும், அதனதன் துவக்க மதிப்புகளை பெற்றுக் கொள்ளச் செய்யும். ஒரு *session*-ஐ உருவாக்கி, அதன் மூலம் இந்த *function*-ஐ நாம்

இயக்க வேண்டும். பின்னர் *for loop* மூலம் *5000* சுற்றுகளை உருவாக்கி, ஒவ்வொரு சுற்றிலும் *cost* கண்டுபிடிப்பதற்கான நிரலையும், *gradient descent* மூலம் அதைக் குறைப்பதற்கான நிரலையும் இயக்க வேண்டும்.அவ்வாறே *X, Y* -க்கான *placeholders*-ஐ வரையறுத்துள்ள இடத்தில், *X_train* மற்றும் *Y_train* -ல் உள்ள தரவுகளை *feed_dict = {}* மூலமாக நாம் செலுத்தியுள்ளோம். இத்தரவுகளுக்கான *cost, 5000* முறை கண்டுபிடிக்கப்பட்டு ஒவ்வொரு முறையும் *gradient descent* மூலம் குறைக்கப்படுகிறது. குறைக்கப்பட்ட இழப்பு *c* எனும் *variable*-ல் *[GD, cost]* என கொடுக்கப்பட்டுள்ளதற்கு ஏற்ப *[None,0.6931285163990998]* எனும் முறையில் சேமிக்கப்படுகிறது. இதில் வலப்பக்கம் உள்ள *cost*-ஐ மட்டும் சேமிக்க *sess.run()[1]* என கொடுக்கப்பட்டுள்ளது. அடுத்ததாக *1000* சுற்றுகளுக்கு ஒருமுறை *cost*-ஐ பிரிண்ட் செய்யுமாறு *if condition* மூலம் கூறியுள்ளோம்.

உருவாக்கியுள்ள நியூரல் நெட்வொர்கை மதிப்பீடு செய்தல்:

எவ்வளவு கணிப்புகள் சரியாக நிகழ்ந்துள்ளன என்பதற்கான டென்சார் correct_prediction எனும் பெயரில் உருவாக்கப்பட்டுள்ளது. அவைகளின் சராசரிக்கான டென்சார் accuracy எனும் பெயரில் உருவாக்கப்பட்டுள்ளது. அதன் மீது செயல்படும் eval() function, பயிற்சித் தரவுகளையும், சோதனைத் தரவுகளையும் palceholders-க்கான இடத்தில் செலுத்தி, இரண்டின் accuracy-ஐயும் கண்டுபிடிக்கிறது.

<u>நிரலுக்கான வெளியீடு:</u>



cost after 0 epoch:

0.6931285163990998

cost after 1000 epoch:

0.05162979674770358 5

…………………………….

cost after 4000 epoch:

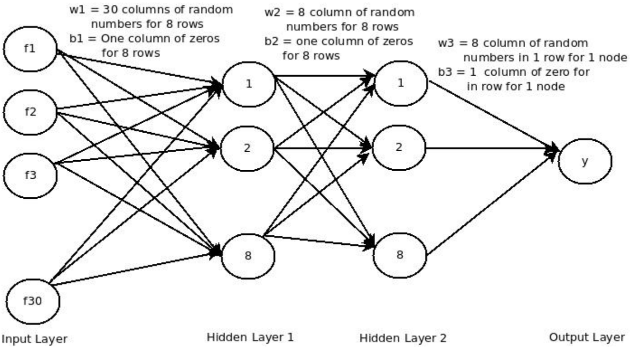
0.0307175769807556

Accuracy for training data: 0.9906103

Accuracy for test data: 0.91608393

8 Deep Neural Networks

ஒன்றுக்கும் மேற்பட்ட hidden layers-ஐ உருவாக்கிக் கற்கும் நெட்வொர்க் deep நியூரல் நெட்வொர்க் அல்லது multi-layer நியூரல் நெட்வொர்க் என்று அழைக்கப்படுகிறது. Shallow-ல் நாம் ஏற்கெனவே பயன்படுத்திய எடுத்துக்காட்டு வழியாக இப்போது இதைக் கற்போம். சென்ற எடுத்துக்காட்டில் நாம் பயன்படுத்திய அனைத்து நிரலையும் இங்கும் பயன்படுத்தியுள்ளோம்.. ஒவ்வொரு லேயருக்குமான அளவுருக்களை வரையறுக்கும் இடத்திலும், அவை கணிப்புகளை நிகழ்த்தும் இடத்திலும் மட்டும் நிரல் வித்தியாசப்படுகிறது.



https://gist.github.com/nithyadurai87/9d931f6b833bce58d6c2b0bdeedd266a

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import
load_breast_cancer
from sklearn.model_selection import
train_test_split

def normalize(data):
```

```
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min,
col_max - col_min)

(X_cancer, y_cancer) =
load_breast_cancer(return_X_y =
True)

X_train, X_test, Y_train, Y_test =
train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = normalize(X_train).T
Y_train = Y_train.reshape(1,
len(Y_train))

X_test = normalize(X_test).T
Y_test = Y_test.reshape(1,
len(Y_test))
```

```
X = tf.placeholder(dtype =
tf.float64, shape =
([X_train.shape[0],None]))
Y = tf.placeholder(dtype =
tf.float64, shape = ([1,None]))

layer_dims =
[X_train.shape[0],8,8,1]
parameters = {}
for i in range(1,len(layer_dims)):
  parameters['W' + str(i)] =
tf.Variable(initial_value=tf.random_
normal([layer_dims[i], layer_dims[i-
1]], dtype=tf.float64)* 0.01)
  parameters['b' + str(i)] =
tf.Variable(initial_value=tf.zeros([
layer_dims[i],1],dtype=tf.float64) *
0.01)

A = X
L = int(len(parameters)/2)
for i in range(1,L):
  A_prev = A
```

```
  Z =
tf.add(tf.matmul(parameters['W' +
str(i)], A_prev), parameters['b' +
str(i)])
  A = tf.nn.relu(Z)
Z_final =
tf.add(tf.matmul(parameters['W' +
str(L)], A), parameters['b' +
str(L)])

cost =
tf.reduce_mean(tf.nn.sigmoid_cross_e
ntropy_with_logits(logits=Z_final,la
bels=Y))

GD =
tf.train.GradientDescentOptimizer(0.
1).minimize(cost)

with tf.Session() as sess:

sess.run(tf.global_variables_initial
izer() )
```

```
  for i in range(5000):
    c = sess.run([GD, cost],
feed_dict={X: X_train, Y: Y_train})
[1]
    if i % 1000 == 0:
      print ("cost after %d
epoch:"%i)
      print(c)
  correct_prediction =
tf.equal(tf.round(tf.sigmoid(Z_final
)), Y)
  accuracy =
tf.reduce_mean(tf.cast(correct_predi
ction, "float"))
  print("Accuracy for training
data:", accuracy.eval({X: X_train,
Y: Y_train}))
  print("Accuracy for test data:",
accuracy.eval({X: X_test, Y:
Y_test}))
```

நிரலுக்கான விளக்கம்:

மேற்கண்ட எடுத்துக்காட்டில் பயன்படுத்திய மார்பகப் புற்றுநோய்க்கான மாதிரித் தரவுகள் *train_test_split* மூலம் பிரிக்கப்படுவதும், பின்னர் அவை *normalize* செய்யப்பட்டு *reshape* செய்யப்படுவதும், உள்ளீட்டு தரவுகளை நியூரல் நெட்வொர்க்குக்குச் செலுத்துவதும், *cost* கண்டுபிடிப்பது, *gradient descent* மூலம் குறைந்த *cost* கண்டறிவது, கடைசியாக *accuracy* கண்டுபிடிப்பது போன்ற அனைத்து இடங்களிலும் *shallow* மற்றும் *deep* நியூரல் நெட்வொர்க் ஒரே மாதிரியாகவே செயல்படுகிறது. ஆனால் பின்வரும் ஒருசில இடங்களில் மட்டும் வித்தியாசப்படுகிறது.

1. Shallow மற்றும் *deep* நியூரல் நெட்வொர்க்கில் முதலாவது லேயர் உள்ளீட்டுக்கானதாகவும், கடைசி லேயர் வெளியீட்டுக்கானதாகவும் அமையும். ஆனால் இடையில் உள்ள *hidden* லேயரைப் பொறுத்து இரண்டும் மாறுபடும். *shallow*-ல் ஒரே ஒரு *hidden* லேயர் மட்டும் காணப்படும். *deep*-ல் ஒன்றுக்கும் மேற்பட்ட லேயர்கள் அமைந்திருக்கும். எனவே *deep*-ல்

ஒவ்வொரு லேயரிலும் உள்ள *nodes*-க்கான அளவுருக்களை வரையறுப்பதற்கு முன்னர், மொத்தம் எத்தனை லேயர்கள் உள்ளன? அவற்றில் எத்தனை *nodes* உள்ளன? என்பதை நாம் வரையறுக்க வேண்டும். இதற்காக இங்கு, *layer_dims = [X_train.shape[0],8,8,1]* என கொடுக்கப்பட்டுள்ளது. முதலாவது லேயர் உள்ளீட்டுக்கானது என்பதால், பயிற்சித் தரவுகளில் உள்ள *features*-ன் எண்ணிக்கையை அளிப்பதற்காக இங்கு *X_train.shape[0]* என கொடுக்கப்பட்டுள்ளது. கடைசியாக உள்ள *1* என்பது வெளியீட்டு லேயரில் உள்ள *nodes*-ன் எண்ணிக்கை ஆகும். இடையில் உள்ள *8,8* என்பதே இரண்டு *hidden* லேயர்கள் உருவாகப்போவதையும், ஒவ்வொரு லேயரிலும் *8 nodes* அமைந்திருப்பதையும் குறிக்கிறது.

layer_dims = [X_train.shape[0],8,8,1]

2. அடுத்ததாக *Shallow*-ல் ஒரே ஒரு *hidden* லேயரில் உள்ள *8 nodes*-க்கான அளவுருக்களை *w1, b1* எனும் பெயரிலும், வெளியீட்டு லேயரில் உள்ள *1 node*-க்கான அளவுருக்களை *w2, b2* எனும் பெயரிலும் அமைத்தோம். தற்போது *deep*-ல் முதல் *hidden* லேயருக்காக *w1, b1* எனும் பெயரிலும், இரண்டாவது *hidden* லேயருக்காக *w2, b2* எனும் பெயரிலும், வெளியீட்டு லேயருக்காக *w3, b3* எனும் பெயரிலும் அளவுருக்களை அமைக்கப்போகிறோம். இந்த *6* அளவுருக்களும் *for loop* மூலம் உருவாக்கப்பட்டு *parameters* எனும் பெயரில் *dictionary* வடிவில் சேமிக்கப்படுகின்றன. இதில் *w1* என்பது இரண்டாவது லேயருக்காக வரையறுக்கப்படுவது. இது முதலாவது லேயரிலிருந்து வரும் *30 features*-க்கான *8 rows*-ஐக் கொண்டிருக்கும். அவ்வாறே அடுத்தடுத்த லேயரில் உள்ள *w2 , w3* ஆகியவை அதற்கு முந்தைய லேயரிலிருந்து வரும் மதிப்புக்களுக்கான அளவுருக்களை கொண்டிருக்கும்.எனவே தான் *layer_dims[i], layer_dims[i-1]* என கொடுக்கப்பட்டுள்ளது.

layer_dims[i] என்பது *weights* அணியில் உள்ள *row*-ன் மதிப்பையும், *layer_dims[i-1]* என்பது அணியில் உள்ள *column*-ன் மதிப்பையும் குறிக்கிறது. முதலாவது லேயருக்கு எந்த ஒரு அளவுரு மதிப்பையும் வரையறுக்கத் தேவையில்லாததால், *for loop*-ஆனது *1* முதல் *len(layer_dims)* வரை அமைவதைக் காணலாம்.

```
parameters = {}

for i in range(1,len(layer_dims)):

  parameters['W' + str(i)] = tf.Variable(initial_value=tf.random_normal([layer_dims[i], layer_dims[i-1]], dtype=tf.float64)* 0.01)

  parameters['b' + str(i)] = tf.Variable(initial_value=tf.zeros([layer_dims[i],1],dtype=tf.float64) * 0.01)
```

3. Shallow-ல் உள்ள ஒரே *hidden* லேயர் தனக்கு முந்தைய லேயரிலிருந்து வரும் *features* மதிப்பையும் *weights* மதிப்பையும் சேர்த்து *Z1, A1* எனும் மதிப்பினைக் கணக்கிடும். கடைசியில் உள்ள வெளியீட்டு லேயர் *Z2* எனும் மதிப்பினை மட்டும் கணக்கிட்டு *activate* செய்யாமல் *cost*-க்குச் செலுத்தும் என்று ஏற்கெனவே பார்த்தோம். *deep* நியூரல் நெட்வொர்க் என்று வரும்போது முதல் *hidden* லேயருக்கான *Z, A* மதிப்புகளையும், இரண்டாவது *hidden* லேயருக்கான *Z, A* மதிப்புகளையும், கடைசி வெளியீட்டு லேயருக்கான *Z* மதிப்பினையும் கணக்கிட வேண்டும். இதற்காக *for loop* ஒன்று செயல்பட்டு முதன் முதலில் *A_prev* எனும் *variable*-ல் *X*-ல் உள்ள *features*-ன் மதிப்பு அமைக்கப்பட்டு முதல் லேயருக்கான *Z , A* மதிப்புகள் கணக்கிடப்படுகின்றன. பின்னர் முதல் லேயர் கணக்கிட்ட *Z* மதிப்பே *A_prev* மதிப்பாக அமைகிறது. இதை *features*-ஆக வைத்து அடுத்த லேயருக்கான *Z, A* மதிப்புகள் கணக்கிடப்படுகின்றன. கடைசி லேயருக்கான *Z* மதிப்பு *loop* முடிந்த பின் கணக்கிடப்படுகிறது.

```
A = X

L = int(len(parameters)/2)

for i in range(1,L):

  A_prev = A

  Z = tf.add(tf.matmul(parameters['W' + str(i)], A_prev), parameters['b' + str(i)])

  A = tf.nn.relu(Z)

Z_final = tf.add(tf.matmul(parameters['W' + str(L)], A), parameters['b' + str(L)])
```

நிரலுக்கான வெளியீடு:

cost after 0 epoch:

0.6931471575913913

cost after 1000 epoch:

0.6641999475918181

cost after 2000 epoch:

0.6641800975355493

cost after 3000 epoch:

0.2876489563036946

cost after 4000 epoch:

0.04967108208362405

Accuracy for training data: 0.9906103

Accuracy for test data: 0.91608393

9 Feed forward neural networks

உள்ளீடு, வெளியீடு மற்றும் பல்வேறு இடைப்பட்ட மறைமுக அடுக்குகளைப் பெற்று, ஒவ்வொரு அடுக்கிலும் அதிக அளவிலான நியூரான்களைப் பெற்று விளங்கும் நெட்வொர்க் feed forward நியூரல் நெட்வொர்க் என்று அழைக்கப்படுகிறது. மேற்கண்ட deep layer-ல் நாம் கண்டது இதற்கு ஒரு நல்ல எடுத்துக்காட்டாக அமையும். உள்ளீட்டு அடுக்கின் மூலம் செலுத்தப்படும் தரவுகள் அதற்கு அடுத்தடுத்த அடுக்குகளின் வழியே செல்லும்போது, அவை process செய்யப்பட்டு கடைசி அடுக்கில் தன்னுடைய கணிப்பினை வெளியிடுகிறது. இம்முறையில் தரவுகள் அனைத்தும் முன்னோக்கி மட்டுமே செலுத்தப்படுகின்றன. பின்னூட்டம் (feedback)

என்ற ஒன்றை எந்த ஒரு அடுக்கும் வழங்குவதில்லை. அதாவது தரவுகளைப் பெற்றுக்கொண்ட நியூரான், தரவுகளை வழங்கிய நியூரானுக்கு பின்னூட்டச் செய்தி அனுப்புவது போன்ற நிகழ்வு ஏதும் இங்கு நடைபெறவில்லை. எனவேதான் இது *feed forward neural network* என்று அழைக்கப்படுகிறது.

கடைசி வெளியீட்டு அடுக்கில் இது கணித்த மதிப்பிற்கும், உண்மையான மதிப்பிற்குமான வேறுபாடு அதிகமாக இருக்கும் பட்சத்தில் அதனுடைய *gradient* மூலம் சற்று பின்னோக்கிப் பயணித்து ஒவ்வொரு அடுக்கிலும் ஏற்கெனவே பயன்படுத்திய அளவுருக்களை மாற்றி அமைக்கும் நிகழ்வு *back propagation* என்று அழைக்கப்படுகிறது. பின்னர் மாற்றப்பட்ட அளவுருக்களுக்கு தரவுகள் அனைத்தும் முன்னோக்கிச் சென்று மீண்டும் கணிப்பினை நிகழ்த்துகின்றன. இவ்வாறே *ffnn* செயல்படுகிறது. இதில் ஒவ்வொரு அடுக்கிலும் உள்ள நியூரான்களின் எண்ணிக்கை குறிப்பிட்ட அளவுக்குக் குறைவாக இருந்தால் *under-fitting*

என்ற பிரச்சனையும், அளவுக்கு அதிகமாக இருந்தால் *over-fitting* என்ற பிரச்சனையும் ஏற்படுகிறது. *over-fitting* என்ற பிரச்சனையைத் தவிர்ப்பதற்காக வந்ததே *drop-out otimization* ஆகும். பொதுவாக அதிக அளவு நியூரான்களைக் கொண்ட நெட்வொர்கில் இதனைப் பயன்படுத்துவதே சிறப்பாக அமையும். அதன் *accuracy*-ஐ அதிகரிக்க உதவும். இதைப்பற்றி *Regularization* மற்றும் *optimization* எனும் பகுதியில் விளக்கமாகக் காணலாம்.

Different Classifiers:

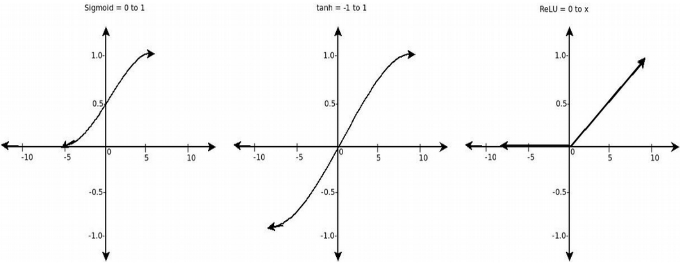
feed forward நியூரல் நெட்வொர்கின் இடைப்பட்ட மறைமுக அடுக்குகளில் *ReLU* எனும் வகைப்படுத்தியும், கடைசி அடுக்கில் *sigmoid* எனும் வகைப்படுத்தியும் பயன்படுத்தப்படுகின்றன. எதற்காக இவ்வாறு வெவ்வேறு வகைப்படுத்திகள் பயன்படுத்தப்படுகின்றன? அவற்றுக்கான தேவைகள் என்ன என்பதைப் பற்றியெல்லாம் இங்கு காணலாம்.

- *sigmoid* - இது தனது கணிப்புகளை *0* முதல் *1* வரை அமைக்கும். கீழ்க்கண்ட வரைபடத்தில் *x*-ன் மதிப்பைப் பொறுத்து கணிக்கப்படும் *h(x)*, *0*-முதல் *1*-வரை அமைய வேண்டுமெனில் அதற்கான சமன்பாடானது *1/(1+e**-x)* என்று இருக்கும். இதுவே *sigmoid function*-க்கான சமன்பாடாக அமைகிறது.

- *Tanh* - இது தனது கணிப்புகளை *-1* முதல் *1* வரை அமைக்கும். இவ்வாறு அமைப்பதற்கான சமன்பாடானது *[2/(1+e**-2x)]-1* என்று இருக்கும். கடைசி வெளியீட்டு லேயரில் *sigmoid* அல்லது *tanh* போன்ற இரண்டில் ஏதாவது ஒன்றைப் பயன்படுத்தலாம்.

- *ReLU* - இதன் கணிப்பு *0* அல்லது அம்மதிப்பாகவே அமையும். *Rectified Linear Unit* என்பதே *Relu* என்றழைக்கப்படுகிறது. இதில் *x*-ன் மதிப்பு *0*-க்குக் கீழ் இருந்தால், அதனை *0*

எனவும், மேல் அமைந்தால் அம்மதிப்பினை அப்படியேயும் கணிக்கிறது. எனவேதான் இதன் சமன்பாடு $max(0,x)$ என அமைகிறது.



Machine learning-ன் அறிமுகக் கற்றலில் வேண்டுமானால், வெறும் *sigmoid*-ஐப் பயன்படுத்தி *logistic regression, neural networks* போன்றவற்றைச் செய்து பார்க்கலாம். ஆனால் *deep* நியூரல் நெட்வொர்க் என்று வரும்போது இதையே அனைத்து அடுக்குகளில் உள்ள நியூரான்களுக்கும் செயல்படுத்துவது சரிவராது. ஏனெனில் மறைமுக அடுக்குகளில் உள்ள நியூரான்களின் மீது *back propagation* என்ற ஒன்றை நாம் செயல்படுத்துவோம். இது ஒவ்வொரு அடுக்கிலும் உள்ள நியூரான்களின் *derivative* மதிப்பைக் கண்டுபிடித்து அதற்கேற்றார்போன்று *weights*-ஐ *update* செய்யும்.

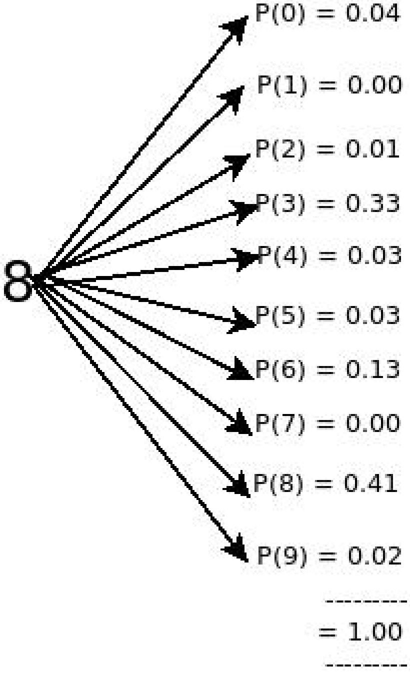
அச்சமயத்தில் வெறும் sigmoid-ஐப் பயன்படுத்தினால், அது வெறும் 0 அல்லது 1 எனும் மதிப்பையே கணிப்பதால், x-ன் மதிப்பைக் கணிசமாகக் குறைக்க உதவாது. எனவேதான் hidden layer-ல் மட்டும் 0 அல்லது அந்த x-ன் மதிப்பையே கணிக்கக்கூடிய ReLU-வை நாம் பயன்படுத்துகின்றோம். கடைசி லேயரில் மட்டும் sigmoid பயன்படுத்தப்பட்டுள்ளது.

10 Softmax neural networks

Softmax என்பது multi-class classification-க்கு உதவுகின்ற ஒரு வகைப்படுத்தி ஆகும். MNIST_data என்பதற்குள் பல்வேறு விதங்களில் கையால் எழுதப்பட்ட 0 முதல் 9 வரை அடங்கிய எண்களின் தொகுப்புகள் காணப்படும். இது 0 - 9 எனும் 10 வகை label-ன் கீழ் அமையக்கூடிய கணிப்புகளை நிகழ்த்தும். இவற்றையே multi-class classification-க்கு மாதிரித் தரவுகளாக நாம் பயன்படுத்திக் கொள்ளலாம். இவற்றில் 55000 தரவுகள் பயிற்சி அளிப்பதற்கும், 10000 தரவுகள் பயிற்சி பெற்ற நெட்வொர்கை சோதிப்பதற்கும் பயன்படுத்தப்படுகின்றன.

Softmax என்பது ஒரு விஷயம் கொடுக்கப்பட்ட ஒவ்வொரு வகையின் கீழும்

கணிக்கப்படுவதற்கான சாத்தியக் கூறுகளை அளந்து கூறும். கீழ்க்கண்ட எடுத்துக்காட்டில், softmax() என்பது ஒரு எண் 0-ஆக அமைவதற்கான சாத்தியக்கூறு எவ்வளவு, 1-ஆக அமைவதற்கான சாத்தியக்கூறு எவ்வளவு என்பது போன்று



மொத்தம் 10 வகை class-க்குமான சாத்தியக்கூறுகளை வகுத்துச் சொல்லும். இந்த சாத்தியக்கூறுகள் 0 முதல் 1 வரை அமைந்த எண்களாகவே இருக்கும். இப்பத்து வரிசைகளில் அமைந்த எண்களையும் கூட்டினால் 1 என வரும். எடுத்துக்காட்டாக கையால் எழுதப்பட்ட 8 எனும் எண் ஒவ்வொரு வகையின் கீழும் கணிக்கப்படுவதற்கான மதிப்பும், அவைகளின் கூட்டுத்தொகை 1 என அமைவதும் பின்வருமாறு அமையும். இவைகளில் எதன் மதிப்பு அதிகமாக இருக்கிறதோ அந்த வகையின் கீழ் கொடுக்கப்பட்ட எண் கணிக்கப்படும்.

https://gist.github.com/nithyadurai87/972397d903915773bc94357942969807

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from
tensorflow.examples.tutorials.mnist
import input_data
from random import randint

mnist =
input_data.read_data_sets("MNIST_dat
a/data", one_hot=True)
print (mnist.train.images.shape)
print (mnist.train.labels.shape)
print (mnist.test.images.shape)
print (mnist.test.labels.shape)
```

```
X = tf.placeholder(dtype =
tf.float32, shape = (None,784),
name='input')
Y = tf.placeholder(dtype =
tf.float32, shape = ([None,10]))

W =
tf.Variable(initial_value=tf.zeros([
784, 10]), dtype = tf.float32)
b =
tf.Variable(initial_value=tf.zeros([
10], dtype=tf.float32))

XX = tf.reshape(X, [-1, 784])
Z = tf.nn.softmax(tf.matmul(XX, W) +
b, name="output")

cost = -tf.reduce_mean(Y *
tf.log(Z)) * 1000.0
```

```
GD =
tf.train.GradientDescentOptimizer(0.
005).minimize(cost)
correct_prediction =
tf.equal(tf.argmax(Z,
1),tf.argmax(Y, 1))
accuracy =
tf.reduce_mean(tf.cast(correct_predi
ction,tf.float32))

with tf.Session() as sess:

sess.run(tf.global_variables_initial
izer())
    writer =
tf.summary.FileWriter('log_mnist_sof
tmax',graph=tf.get_default_graph())
    for epoch in range(10):
        batch_count =
int(mnist.train.num_examples/100)
        for i in range(batch_count):
            batch_x, batch_y =
mnist.train.next_batch(100)
```

```
            c = sess.run([GD, cost],
feed_dict={X: batch_x, Y: batch_y})
[1]
        print ("Epoch: ", epoch)
    print ("Accuracy: ",
accuracy.eval(feed_dict={X:
mnist.test.images,Y:
mnist.test.labels}))
    print ("done")

    num = randint(1,
mnist.test.images.shape[1])
    img = mnist.test.images[num]

    classification =
sess.run(tf.argmax(Z, 1),
feed_dict={X: [img]})
    print('Neural Network
predicted', classification[0])
    print('Real label is:',
np.argmax(mnist.test.labels[num]))
```

(55000, 784)

(55000, 10)

(10000, 784)

(10000, 10)

Epoch: 0

Epoch: 1

Epoch: 2

Epoch: 3

Epoch: 4

Epoch: 5

Epoch: 6

Epoch: 7

Epoch: 8

Epoch: 9

Accuracy: 0.9226

done

Neural Network predicted 3

Real label is: 3

11 Building Effective Neural Networks

ஒரு நியூரல் நெட்வொர்கின் கட்டமைப்பினை வலுவாக்குவதற்கு முதலில் அதில் எழுகின்ற பல்வேறு வகையான பிரச்சனைகள் பற்றியும், அவற்றைக் களைவதற்கு உதவும் வழிவகைகள் பற்றியும் தெரிந்து கொள்ள வேண்டும்.. ஒரு நியூரல் நெட்வொர்கின் efficiency என்பது பயிற்றுவிக்கப்படுவதற்கு எவ்வளவு நேரம் எடுத்துக்கொள்கிறது என்பதையும், Accuracy என்பது பயிற்றுவிக்கப்பட்டவுடன் எவ்வளவு துல்லியமாகக் கணிக்கிறது என்பதையும் குறிக்கிறது. இவற்றைப் பாதிக்கும் காரணிகள் பற்றியும், அவற்றை எவ்வாறு களைவது என்பது பற்றியும் இப்பகுதியில் ஒவ்வொன்றாகப் பார்க்கலாம்.

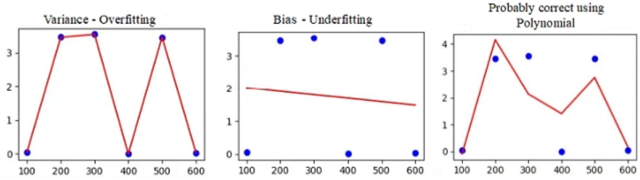
.1 Bias-Variance Problem

நாம் உருவாக்கியுள்ள நெட்வொர்க் பயிற்சியின் போது அதிக அளவு *accuracy*-ஐ வெளிப்படுத்தினாலும், நிஜத்தில் அதனால் திறம்பட செயல்பட்டு சரிவர கணிக்க முடியவில்லையெனில், அதற்கு *bias* அல்லது *variance* பிரச்சனை நிகழ்ந்துள்ளது என்று அர்த்தம்.

மாதிரித் தரவுகளில் உள்ள *features*-ன் எண்ணிக்கை தரவுகளின் எண்ணிக்கையை விட மிக மிகக் குறைவாக இருக்கும்போது *bias* எனும் பிரச்சனை ஏற்படுகிறது. ஏனெனில் ஓரிரண்டு *features*-ஐ வைத்து மட்டுமே அதிக அளவில் அமைந்துள்ள தரவுகளைப் பற்றிக் கற்றுக் கொள்ள முயல்கிறது. இவ்வாறாக கற்றல் என்பது பரவலாக அமையாமல், ஓரிரண்டு *features*-ஐப் பொறுத்து மட்டுமே அமைவது *bias / under fitting* என்று அழைக்கப்படுகிறது.

அதே போன்று மாதிரித் தரவுகளில் உள்ள *features*-ன் எண்ணிக்கை தரவுகளின் எண்ணிக்கையை விட மிக மிக அதிகமாக இருக்கும் பட்சத்தில், *variance / over fitting* என்ற பிரச்சனை ஏற்படுகிறது. இதற்கான காரணங்களாக இரண்டு விஷயத்தைக் கூறலாம். முதலில் தரவுகளின் எண்ணிக்கை குறைவாக இருப்பதால், அவற்றை சுலபத்தில் மனப்பாடம் செய்துவிடுகிறது. அடுத்து *features*-ன் எண்ணிக்கை அதிக அளவில் இருப்பதால், சற்றும் சம்பந்தமில்லாத *features*-யிடம் இருந்து கூட ஏதோ ஒன்றைக் கற்றுக் கொண்டு அதன்வழி செயல்பட ஆரம்பிக்கிறது.

இவைகளுக்கான வரைபடம் கீழ்க்கண்டவாறு அமையும். இதை எளிய தமிழில் *Machine Learning* என்ற புத்தகத்தில் காணலாம். இதை உருவாக்குவதற்கான நிரலும் அப்புத்தகத்தில் *'Polynomial Regression'* என்ற பகுதியில் கொடுக்கப்பட்டுள்ளது.



இத்தகைய *bias-variance* பிரச்சனையைத் தீர்க்க தரவுகளை நாம் *training, develop, testing* என மூன்றாகப் பிரித்துக் கொள்ளலாம். இதன் மூலம் நாம் உருவாக்கியுள்ள மாடல் எந்த வகைப் பிரச்சனைக்கு ஆளாகியுள்ளது என்பதையும் கண்டுபிடிக்கலாம். *training*-க்கான தரவுகளை வைத்து பயிற்சி அளித்து மாடலை உருவாக்கலாம். *develop*-க்கான தரவுகளை வைத்துக் கொண்டு பரிசோதித்து ஒருசில *hyper-parameters* ஐ வேண்டியவாறு மாற்றி அமைத்துக் கொள்ளலாம். பின்னர் *testing*-க்கான தரவுகள் மூலம், மாடலை பரிசோதித்து மதிப்பிடலாம். இவை ஒவ்வொன்றும் வெளிப்படுத்தும் *cost*

மதிப்புகளை வைத்துக் கொண்டு மாடல் எந்த வகைப் பிரச்சனைக்கு ஆளாகியுள்ளது என்பதைக் கண்டுபிடித்து விடலாம். எடுத்துக்காட்டாக,

1. *Training error 80% -*க்கும் அதிகமாக இருந்தால் *bias* பிரச்சனை என்றும்,

2. *Training error* குறைவாகவும் *(2%), Dev error* அதிகமாகவும் *(30%)* இருந்தால் *variance* பிரச்சனை என்றும்,

3. *Training* மற்றும் *Dev* இரண்டிலும் *error* மதிப்பு குறைவாகவும், *Testing error* மதிப்பு அதிகமாகவும் இருந்தால், *variance* பிரச்சனை *training & dev* இரண்டிலும் நிகழ்ந்துள்ளது என்று அர்த்தம்.

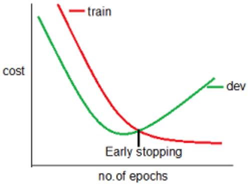
Early Stopping என்பது இப்பிரச்சனையைத் தீர்க்க உதவும் ஒரு வழிவகை ஆகும். *Normalization*

என்பது *bias* பிரச்சனையையும், *Regularization* என்பது *variance* பிரச்சனையையும் தீர்க்க உதவுகிறது. *weights decay, drop-out* போன்றவை நியூரல் நெட்வொர்கில் மட்டும் பயன்படுத்தப்படும் சிறப்பு வகை *regularization* நுட்பங்கள் ஆகும்.

..1 Early Stopping

Gradient Descent மூலம் *error* மதிப்பு கூடுகை அடைய முயலும்போது, *Training* மற்றும் *Dev* இரண்டும் வெளிப்படுத்துகின்ற *cost* மதிப்பை ஒரு வரைபடமாக வரைந்து பார்க்கவும். *Training* மதிப்பு குறைந்து கொண்டே வந்தால் அதோடு சேர்ந்து *dev* மதிப்பும் குறைந்து கொண்டே வர வேண்டும். ஏதாவதொரு நிலையில் *dev* மதிப்பு அதிகரிக்கத் தொடங்கினால், சுழற்சியை அந்நிலையிலேயே நிறுத்திவிட்டு அதன் அளவுருக்களை நமது மாடலுக்குப்

பயன்படுத்திக் கொள்ளலாம். இதுவே *'Early Stopping'* என்று அழைக்கப்படுகிறது.



அதாவது கூடுகை *(Convergence)* அடைவதற்கு முன்னரே சுழற்சி நிறுத்தப்பட்டு அளவுருக்கள் தேர்ந்தெடுக்கப்படுவதால் இது இப்பெயரில் அழைக்கப்படுகிறது.

..2

..3 Normalization

ஒவ்வொரு *feature column*-ல் உள்ள மதிப்புகளும் வெவ்வேறு எண் எல்லைகளில்

அமைந்திருந்தால், அனைத்தையும் -1 லிருந்து +1 வரை, அல்லது 0 லிருந்து 1 வரை என சீராக்குவதே Normalization அல்லது feature scaling எனப்படும். பின்வரும் சூத்திரத்தைப் பயன்படுத்தி சீராக்கப்பட்டால் அதுவே mean normalization எனப்படும். இதைப் பற்றி இன்னும் தெளிவாக அறிய விரும்பினால், எளிய தமிழில் Machine Learning என்ற புத்தகத்தில் 'Multiple Linear Regression' என்ற பகுதியில் காணவும்.

$$\frac{\text{particular value} - \text{mean of all values}}{\text{maximum} - \text{minimum}}$$

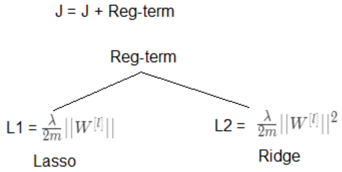
..4 L1 , L2 Regularization

மிகக் குறைந்த அளவு தரவுகளுக்கு, அதிக அளவு பங்களிப்பதால் ஏற்படுவதே பிரச்சனை என்று பார்த்தோம். இதைத் தவிர்க்க ஒன்று அதிக அளவு தரவுகளைக் கொண்டு வரலாம் அல்லது ஒவ்வொரு feature-ன் பங்களிப்பையும் குறைந்த அளவில் மாற்றலாம். இதில் அதிக அளவு தரவுகளைக் கொண்டு வருவது மிகுந்த நேரமும்

செலவும் பிடிக்கும் வேலையாதலால் ஒவ்வொரு *feature*-ன் பங்களிப்பையும் குறைக்க அதனுடன் *regularization*-க்கான அளவீடு இணைக்கப்படுகிறது.

இத்தகைய நெறிப்படுத்தும் முறையானது *L1* மற்றும் *L2* எனும் இரண்டு நிலைகளில் நடைபெறுகிறது. மிக மிக அதிக அளவில் *features* இருக்கும்போது *L1* வகை பயன்படுத்தப்படுகிறது. இது அதிக முக்கியமில்லாத *features*-க்கான மதிப்பினை சுழியம் *(0)* என அமைப்பதன் மூலம் முக்கியமானவற்றை மட்டும் பங்களிக்க வைக்கிறது. ஒரு *regression* மாடலானது *L1*-ஐப் பயன்படுத்தினால் அது *Lasso Regression (Least Absolute shrinkage & selection operator)* என்றும், *L2*-ஐப் பயன்படுத்தினால் அது *Ridge regression* என்றும் அழைக்கப்படுகிறது. *L2* என்பது நெட்வொர்கில் உள்ள அனைத்து அளவுருக்களின் கூட்டுத் தொகையையும் இரட்டிப்பாக்கி அதனை லாம்டா */2m* எனும் மதிப்பினால் பெருக்குகிறது. இவ்வாறாக *L1*

மற்றும் L2 -ஆல் கண்டறியப்படும் மதிப்பே 'Penalty term' என்று அழைக்கப்படுகிறது. பின்னர் இம்மதிப்பே cost (J) மதிப்புடன் இணைக்கப்படுகிறது. சுருக்கமாகச் சொல்லப்போனால் L1 என்பது 'Absolute value of Magnitude' என்றும், L2 என்பது 'Squared Magnitude' என்றும் அறியப்படும்.

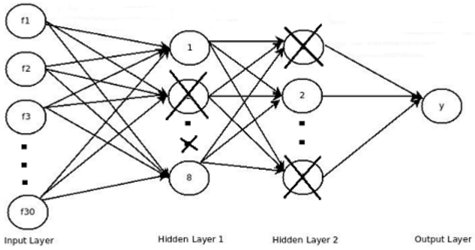


இம்முறைகளில் நாம் வரையறுக்கும் லாம்டாவின் மதிப்பு மிக மிகக் குறைவாக இருந்தால், *overfitting* என்ற பிரச்சனை அப்படியே நிலைத்திருக்கும். மிக மிக அதிகமாக இருந்தால் *underfitting* நிலைக்குச் சென்றுவிடும். எனவே இந்த லாம்டா மதிப்பினை தேர்வு செய்யும்போது, மிக மிகக் கவனமாக இருக்க வேண்டியது அவசியமாகிறது.

..5 Drop-out Regularization

இது ஒவ்வொரு லேயரிலும் தோராயமாக ஒருசில *nodes*-ஐ நீக்கி விடுவதன் மூலம் *features*-

ன் எண்ணிக்கையைக் குறைக்கும். *keep_prob* என்பது இது பயன்படுத்தும் *parameter* ஆகும். இதன் மூலம் ஒவ்வொரு லேயரிலும் எவ்வளவு *nodes* இருக்க வேண்டும் என்பதை வரையறுக்கலாம். அதாவது *keep_prob = 0.6* என்று இருந்தால் *60% nodes*-ஐ வைத்துக் கொண்டு மீதி *40% nodes*-ஐ நீக்கி விடலாம் என்று அர்த்தம். ஒவ்வொரு லேயருக்கும் வெவ்வேறு *keep_prob* மதிப்புகளை நாம் வரையறுக்கலாம்.



.2 Data-Insufficiency

இது பயிற்சி அளிப்பதற்குப் போதுமான தரவுகள் இல்லாமையால் ஏற்படுகின்ற பிரச்சனை ஆகும். இதனைத் தவிர்க்க *Data Augmentation* என்ற முறையைப் பயன்படுத்தலாம். இதற்கு தரவுகளைப் பெருக்குதல் என்று பொருள். அதாவது நம்மிடம் ஏற்கெனவே இருக்கின்ற தரவுகளைக் கொண்டு அதிக அளவில் தரவுகளை உருவாக்கிப் பயன்படுத்தலாம். *Mirroring* (பிரதிபலித்தல்), *Random cropping* (சீரற்ற துண்டுகளாக்குதல்), *Rotation* (சுழற்றுதல்), *Shearing* (வெட்டுதல்), *Colour*

shifting (நிறமாற்றம் செய்தல்) போன்றவை இத்தகைய தரவுகளின் பெருக்கத்திற்கு உதவுகின்ற முறைகள் ஆகும்.

.3 Vanishing & Exploding gradient

நியூரல் நெட்வொர்கில், நாம் பயன்படுத்தியுள்ள *weights*-ன் மதிப்பு மிகவும் சிறியதாக இருந்தால், அதன் *gradients (dw, db) backpropagation*-ன் போது ஒரு கட்டத்தில் தானாக மறைந்துவிடும். இதுவே *vanishing gradients* எனப்படும். இதற்கு நேர் மாறாக *gradients*-ன் மதிப்பு மிகப் பெரியதாக அமைவது *exploding gradients* எனப்படும். *RNN* போன்ற நீண்ட தொடர்புடைய வார்த்தைகளை நினைவில் வைத்துக் கொள்ள வேண்டிய மிகவும் ஆழமான நெட்வொர்குக்கு இத்தகைய பிரச்சனைகள் எழுகின்றன. (*RNN* பற்றி இனிவரும் பகுதிகளில் காணலாம்)

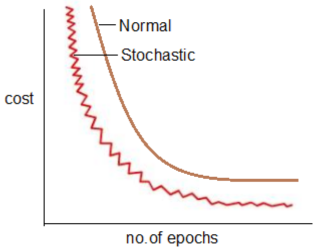
hidden layer-ல் *tanh activation fn*-ஐப் பயன்படுத்துவதன் மூலம் இத்தகைய பிரச்சனைகளைத் தவிர்க்கலாம். மேலும் *Xavier initialization* மூலம் *weights*-க்கான துவக்க மதிப்புகளை அளிக்கும்போது, அது மதிப்புகள் மிகவும் சிறியதாகவும் இல்லாமல், மிகவும் பெரியதாகவும் இல்லாமல் அமைவதை உறுதி செய்கிறது. அதாவது *weights matrix*-க்கான துவக்க மதிப்புகளை அளிக்கும்போது, அதன் *variance = 2 / n (where n = no. of features)* என்று இருக்குமாறு பார்த்துக்கொள்கிறது. *'Gradient Clipping'* என்பது *exploding* பிரச்சனையைத் தவிர்க்க உதவும் ஒரு வழிவகை ஆகும். இது *gradients*-ன் மதிப்புகள் ஒரு குறிப்பிட்ட எல்லையைத் தாண்டிச் செல்லும்போது, அவற்றை எல்லைக்குள் அமைத்து பின்னர் *back propagation* வழியே செலுத்தி *weights* மதிப்புகளை மேம்படுத்துகிறது.

.4 Mini-batch Gradient descent

Gradient descent என்றால் சாய்வு வம்சாவளி என்று பொருள். ஒரு நெட்வொர்க் இம்முறையைப் பயன்படுத்தி கூடுகை(*convergence*) அடைய முயலும்போது ஒவ்வொரு சுழற்சியிலும், முழுத் தரவுகளையும் பயன்படுத் திப் பயிற்சி அளிப்பதால், அதிக அளவு நேரமும் செயல்திறனும் வீணாகிறது. இதனைத் தவிர்ப்பதற்காக வந்ததே *mini-batch gradient descent* ஆகும். இது முழுத் தரவுகளையும் பல்வேறு தொகுதிகளாகப் பிரித்து, ஒவ்வொரு சுழற்சியிலும் ஒரு *batch*-ஐ அனுப்பி அதனடிப்படையில் *gradients* மதிப்புகளை திருத்தம் செய்கிறது. எடுத்துக்காட்டாக பயிற்சித் தரவுகளின் எண்ணிக்கை மொத்தம் பத்தாயிரம் இருக்கிறதெனில், இது ஆயிரம் ஆயிரமாக அமைந்த மொத்தம் 10 தொகுதிகளை உருவாக்கும். பின்னர் ஒவ்வொரு சுழற்சிக்கும் ஒவ்வொரு தொகுதியாக அனுப்பி *gradients*-ஐ திருத்தம் செய்யும். இதுவே *Stochastic Gradient*

descent எனப்படும். அதுவே பிரிக்கப்படும் தொகுதிகளின் எண்ணிக்கை மொத்தத் தரவுகளின் எண்ணிக்கைக்கு சமமாக பத்தாயிரம் என அமைந்தால், அதுவே *Batch Gradient descent* அல்லது *Gradient descent without mini-batch* என்று அழைக்கப்படுகிறது.

Stochastic மற்றும் *Batch* இவ்விரண்டும் சாய்வு வம்சாவளி மூலம் கூடுகை அடைவதை *(Convergence using gradient descent)* வரைபடமாக வரைந்து பார்த்தால், *Stochastic*-ல் கூடுகை என்பது விரைவாக நடைபெற்று விடும். ஏனெனில் *Batch*-ல் *mini batch*-ன் எண்ணிக்கை மொத்த எண்ணிக்கைக்கு சமமாக அமைவதால் கூடுகை என்பது சற்று நேரம் பிடிக்கும். மேலும் *Stochastic*-ன் வரைபடத்தில் காணப்படும் பல்வேறு ஏற்ற இறக்கங்கள் ஒவ்வொரு தொகுதிகளைப் பயன்படுத்தி இறங்கி வருவதை உணர்த்துகிறது.

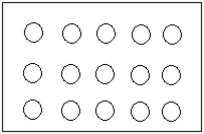


.5 Hyper-parameter Tuning

நியூரல் நெட்வொர்கில் ஒருசில அளவுருக்கள் தானாக தன்னுடைய மதிப்பினைக் கற்றுக்கொள்கின்றன. ஒருசில அளவுருக்களுக்கு மதிப்பினை நாம் வரையறுக்க வேண்டியுள்ளது. எடுத்துக்காட்டாக *weights* , *bias* போன்ற அளவுருக்களுக்கு, துவக்கத்தில் ஒருசில மதிப்புகளை அளித்து நெட்வொர்கிடம் விட்டுவிடுகிறோம். பின்னர் அவை பயிற்சியின்போது தானாக சரியான

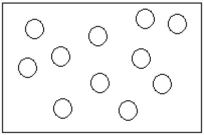
மதிப்புகளைப் பெற்றுக்கொள்கின்றன. ஆனால் ஒரு நெட்வொர்கில் எவ்வளவு அடுக்குகள் இருக்க வேண்டும், அவற்றில் எத்தனை *nodes* இருக்க வேண்டும் என்பதையெல்லாம் நாம்தான் வரையறுக்க வேண்டும். இவ்வாறாக தானாகக் கற்றுக் கொள்ளாமல், நாம் மதிப்பினை வரையறுக்கும் அளவுருக்களுக்கு *hyper-parameters* என்று பெயர். *Layers*-ன் எண்ணிக்கை , *Nodes*-ன் எண்ணிக்கை , *Mini batches*-ன் எண்ணிக்கை , *Gradient descent*-ல் பயன்படுத்தப்படும் கற்றலுக்கான விகிதம்*(learning rate)* போன்றவற்றை இதற்கான எடுத்துக்காட்டாகச் சொல்லலாம். இவற்றை நாம் சரிவர அமைக்கும்வரை பல்வேறு மதிப்புகளை ஒவ்வொன்றாகச் சோதித்துப் பார்த்து சரியானதை தேர்வு செய்வதையே *hyper-parameter tuning* என்கிறோம். இது பல்வேறு முறைகளில் நடைபெறுகிறது. அவை பின்வருமாறு.

5.1 Grid Search



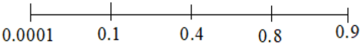
ஒரு அணியில் பல்வேறு அளவுருக்களை அமைத்து அவற்றிலிருந்து ஒவ்வொரு combination-ஆக சோதித்துப் பார்த்து அதிலிருந்து ஒன்றை தேர்வு செய்வதையே Grid search என்கிறோம். தேர்ந்தெடுக்கப்பட வேண்டிய அளவுருக்களின் எண்ணிக்கை குறைவாக இருக்கும்போது இம்முறை சிறப்பாக அமையும்.

5.2 Random Search



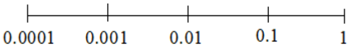
எந்த ஒரு வரையறையும் வைத்துக் கொள்ளாமல், சீரற்ற முறையில் அளவுருக்களைத் தேர்ந்தெடுக்கும் முறைக்கு Random Search என்று பெயர். சீரற்ற முறையில் தேர்ந்தெடுக்கப்பட்டாலும், இதுவே grid search-ஐ விட சிறப்பாக அமையும். தேர்ந்தெடுக்கப்பட வேண்டிய அளவுருக்களின் எண்ணிக்கை அதிகமாக இருக்கும் பட்சத்தில் இம்முறையை நாம் பயன்படுத்தலாம்.

5.3 Linear Scale



லேயர்களின் எண்ணிக்கை, Nodes-ன் எண்ணிக்கை போன்ற குறைவான எண்ணிக்கையில் அமையும் அளவுருக்களைத் தேர்வு செய்ய இம்முறையை நாம் பயன்படுத்தலாம். இதன் மதிப்புகள் non uniform-ஆக அமையும். [Eg: 0.001, 0.1, ... 0.9]

5.4 Log Scale

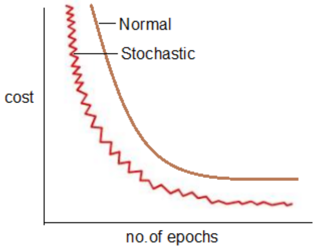


கற்றலுக்கான விகிதம்(learning rate) போன்ற அதிமுக்கிய மதிப்புகளைத் தேர்வு செய்ய இம்முறையைப் பயன்படுத்தலாம். ஏனெனில் இதுபோன்ற மதிப்புகளில் ஒரு சிறிய அளவில் மாற்றம் ஏற்பட்டால் கூட அது மொத்த நெட்வொர்கின் செயல்திறனையும் பாதித்துவிடும். ஆகவே இம்முறையில் மதிப்புகள் uniform-ஆக 0.0001, 0.001, 0.01, 0.1 என்பது போன்று அமைவதால், இதிலிருந்து

ஒவ்வொன்றாகச் சோதித்துப் பார்த்து தேர்வு செய்யலாம்.

.6 Optimization Techniques

நமது நெட்வொர்கின் செயல்திறனை அதிகரிப்பதற்காக சிறந்த *hyperparameters*-ஐத் தேர்வு செய்யும் நோக்கில் பல்வேறு *optimization* வழிவகைகள் பயன்படுத்தப்படுகின்றன. *Stochastic GD* மூலம் விரைவாக கூடுகை அடைய முடிந்தாலும், அது தனக்கான பாதையில் பல்வேறு ஏற்ற இறக்கங்களைக் கொண்டிருப்பதைக் காணலாம்.



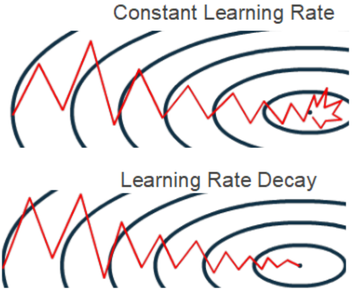
இதனை சரி செய்து சீரான முறையில் கூடுகை அடைவதற்கு உதவ பின்வரும் *optimization* வழிவகைகள் இங்கு பயன்படுத்தப்படுகின்றன. இதன் மூலம் சிறந்த அளவுருக்கள் தேர்ந்தெடுக்கப்படுகின்றன.

- *Gradient Descent with Momentum* என்பது அதிவேகமாக நகரும் சராசரியைக் *(exponential moving average)* கணக்கிட்டுப் பயன்படுத்துவதன் மூலம் இத்தகைய *oscillations*-ஐக் கட்டுப்படுத்த முயல்கிறது.

- *Root Mean Square propagation* என்பதே *RMS Prop* என்றழைக்கப்படுகிறது. அதாவது மூல சராசரி சதுர மதிப்பினைக் கண்டுபிடித்து அதன் மூலம் பரவி, *oscillations*-ஐக் கட்டுப்படுத்தி சிறந்த அளவுருக்களைத் தேர்ந்தெடுப்பதே *RMS Prop* என்றழைக்கப்படுகிறது.

- மேற்கூறிய இரண்டு தத்துவங்களின் கலப்பே *Adam optimization* ஆகும்.

6.1 Learning Rate Decay



ஒரு நெட்வொர்கின் *cost* மதிப்பு *gradient descent* மூலம் கூடுகை அடைய முயலும்போது, அதன் மொத்த சுழற்சிகளிலும் கற்றலுக்கான விகிதம் என்பது மாறாத ஒரு மதிப்பாக அமைகிறது.

எனவே அதன் சுழற்சிகளில் ஒன்றின் போது *cost* மதிப்பு கூடுகைக்கு அருகில் அமைந்தால் கூட, அதன் கற்றலுக்கான மாறாத மதிப்பைப் பொறுத்து சற்று நீண்டு அடுத்த அடியை எடுத்து வைக்கிறது. எனவே கூடுகை சமயத்தில் அதனை விட்டு விலகிச் சென்று அடுத்தடுத்த சுழற்சிகளில் மட்டுமே கூடுகையை அடைகிறது. இவ்வாறு கூடுகை சமயத்தில் தேவையில்லாமல் நிகழ்த்தப்படும் அதிகப்படியான சுழற்சிகளைத் தவிர்ப்பதற்காக வந்ததே *learning rate decay* ஆகும். இது கற்றலுக்கான விகித மதிப்பை ஒவ்வொரு சுழற்சியிலும் கொஞ்சம் கொஞ்சமாக குறைத்துக் கொண்டே வரும். எனவே கூடுகை சமயத்தின் போது தனது அடுத்தடுத்த அடிகளை சின்னஞ் சிறியதாக எடுத்து வைப்பதால் சுலபத்தில் கூடுகையை அடைந்துவிடும். இதனால் தேவையில்லாமல் நிகழும் அதிகப்படியான சுற்றுகள் தவிர்க்கப்படுகின்றன.

6.2 Pruning

இது நெட்வொர்கின் செயல்திறனை அதிகரிக்க உதவும் ஒரு வழிவகை ஆகும். எடுத்தவுடன் நெட்வொர்கை அதிக அளவு நியூரான்களுடன் வடிவமைப்பதால் அதன் செயல்திறன் தொடக்க நிலையிலேயே முடக்கப்படுகிறது. *Pruning* என்பது கொஞ்சம் கொஞ்சமாக நெட்வொர்கில் உள்ள நியூரான்களின் எண்ணிக்கையை அதிகரிக்க உதவும் ஒரு வழிவகை ஆகும். இதன் மூலம் நெட்வொர்கின் செயல்திறன் பாதிக்கப்படாமல் பார்த்துக் கொள்ளலாம்.

6.3 Batch Normalization

Normalization பற்றி எளிய தமிழில் என்ற புத்தகத்தில் *'Multiple linear regression'* எனும் பகுதியில் விளக்கியிருப்பேன். அதாவது ஒரு வீட்டின் விலையானது, அதன் சதுர அடி விவரம் மற்றும் அவற்றிலுள்ள அறைகளின் எண்ணிக்கை எனும் *2 input features*-ஐப் பொறுத்து அமைகிறதெனில் சதுர அடியானது *450-sqft* முதல் *2500-sqft* வரை பரவியிருக்கும். ஆனால் அறைகளின் எண்ணிக்கையோ *1* முதல்

அதிகபட்சம் 5 வரை மட்டுமே பரவியிருக்கும். இவ்வாறு வெவ்வேறு எல்லைகளில் பரவியிருக்கும் input features-வுடைய தரவுகளை சீராக -1 முதல் +1 வரை அமைக்க முயல்வதே Normalization என்று கண்டோம்.

சாதாரண regression-ல் செயல்படும் Normalization-ஐப் போலவே நியூரல் நெட்வொர்கில் செயல்படுவது 'Batch Normalization' ஆகும். ஏனெனில் நெட்வொர்க்கு பயிற்சி அளிக்கச் செலுத்தப்படும் தரவுகள் mini-batch gradient descent மூலம் செலுத்தப்படும் போது அவை பல்வேறு தொகுதிகளில் அமைகின்றன. எனவே ஒவ்வொரு தொகுதியிலும் அமையும் தரவுகள் பல்வேறு எல்லை வரிசைகளில் இருப்பதற்கான வாய்ப்புகள் அதிகம். இதன் விளைவாக ஏற்படுவதே 'co-variance shift' (Change in data distribution of the input features) எனும் பிரச்சனை ஆகும். ஏனெனில் ஒவ்வொரு முறையும் நியூரான்கள் வெவ்வேறு தரவு எல்லைகளுக்கு ஏற்ப தம்முடைய அளவுருக்களை மாற்றி அமைக்க முயலும். இம்முறையில்

பயிற்றுவிக்கப்படுவது சரியாக இராது. எனவே ஒவ்வொரு தொகுதிகளிலும் உள்ள தரவுகள் அடுத்த லேயருக்கு அனுப்பப்படுவதற்கு முன்னர் அவை *normalize* செய்யப்பட்டு அனுப்பப்படுவதே *'Batch Normalization'* என்று அழைக்கப்படுகிறது. நியூரல் நெட்வொர்கில் இவை தொகுதி வாரியாக நடைபெறுவதால் இது இப்பெயரில் அழைக்கப்படுகிறது. *CNN* போன்ற நெட்வொர்குக்கு பயிற்சி அளிக்கும்போது கருப்பு-வெள்ளைப் படங்களைக் கொடுத்து பயிற்சி அளிப்பதும், பின்னர் வண்ணப் புகைப்படங்களைக் கொடுத்து கணிக்கச் சொல்லுவதும் இத்தகைய *'covariance shift'* பிரச்சனைக்கு உதாரணங்களாக அமையும். ஏனெனில் இவற்றின் பிக்சல் மதிப்புகளில் அதிக அளவு வேறுபாடு காணப்படும். (*CNN* பற்றி இனிவரும் பகுதிகளில் விளக்கமாகக் காணலாம்)

.7 Example Program

deep neural network-ல் நாம் பயன்படுத்தியுள்ள மார்பகப் புற்றுநோய்க்கான எடுத்துக்காட்டையே இங்கும் பயன்படுத்தியுள்ளோம். கீழ்க்கண்ட இடங்களில் மட்டும் நிரல் சற்று வித்தியாசப்படுகிறது.

1. மாதிரித் தரவுகள் train_test_split மூலம் பிரிக்கப்பட்ட உடன், X_train-ல் உள்ள முதல் 2 features மட்டும் கணக்கிற்கு எடுத்துக் கொள்ளப்படுகிறது. பின்னர் இவ்விரண்டு தரவுகளும் non-linear எவ்வாறு முறையில் அமைந்துள்ளன என்பது வரைபடம் மூலம் வரைந்து காட்டப்பட்டுள்ளது.

2. hidden லேயரில் உள்ள ஒவ்வொரு node-ம் முந்தைய லேயரிலுள்ள node-லிருந்து வரும்

மதிப்புடன் *weight*-ஐப் பெருக்கி கடைசியாக அவற்றுடன் *bias*-ஐக் கூட்டும் நிகழ்வு *tf.matmul(), tf.add()* மூலம் நடைபெறுகிறது. இவ்வாறு கணக்கிடப்பட்ட மதிப்பே *hidden* லேயரில் *relu* மூலம் *activate* செய்யப்படுகிறது. கடைசி லேயரில் மட்டும் *activate* செய்யப்படாத இம்மதிப்பு, *cost* கண்டுபிடிக்கச் செலுத்தப்படுகிறது. ஏனெனில் *cost*-ன் ஒரு பகுதியாக கடைசி லேயருக்கான *sigmoid activation* நடைபெற்று விடுகிறது என்று ஏற்கெனவே பார்த்தோம். இத்தகைய *forward propagation* முறையானது ஒரு தனி *function*-ஆக எழுதப்பட்டுள்ளது.

இதில் *dropout optimization* என்பது மறைமுக அடுக்குகளில், *relu activation* நடைபெற்ற பிறகு ஒருசில நியூரான்களை, கொடுக்கப்பட்ட முடக்க விகிதத்திற்கு ஏற்றார்போல் செயலிழக்கம் செய்கின்ற முறை ஆகும். இதற்கான நிரல் பின்வருமாறு இணைக்கப்பட்டுள்ளது.

```
if drop_out == True:
```

A = tf.nn.dropout(x = A, keep_prob = 0.8)

[*https://gist.github.com/nithyadurai87/4baef96f858284d1a4868c80df25f990*](https://gist.github.com/nithyadurai87/4baef96f858284d1a4868c80df25f990)

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import
load_breast_cancer
from sklearn.model_selection import
train_test_split
import pandas as pd

def normalize(data):
  col_max = np.max(data, axis = 0)
  col_min = np.min(data, axis = 0)
  return np.divide(data - col_min,
col_max - col_min)
```

```
(X_cancer, y_cancer) =
load_breast_cancer(return_X_y =
True)
X_train, X_test, Y_train, Y_test =
train_test_split(X_cancer, y_cancer,
random_state = 25)

X_train = X_train[:,:2]

import matplotlib.pyplot as plt
import matplotlib.colors
colors=['blue','green']
cmap =
matplotlib.colors.ListedColormap(col
ors)
plt.figure()
plt.title('Non-linearly separable
classes')
plt.scatter(X_train[:,0],
X_train[:,1], c=Y_train, marker=
'o', s=50,cmap=cmap,alpha = 0.5 )
plt.show()
```

```
def forwardProp(X, parameters,
drop_out = False):
    A = X
    L = len(parameters)//2
    for l in range(1,L):
        A_prev = A
        A =
tf.nn.relu(tf.add(tf.matmul(paramete
rs['W' + str(l)], A_prev),
parameters['b' + str(l)]))
        if drop_out == True:
            A = tf.nn.dropout(x = A,
keep_prob = 0.8)
    A =
tf.add(tf.matmul(parameters['W' +
str(L)], A), parameters['b' +
str(L)])
    return A

def deep_net(regularization = False,
lambd = 0, drop_out = False,
optimizer = False):
```

```
    tf.reset_default_graph()
    layer_dims = [2,25,25,1]
    X = tf.placeholder(dtype =
tf.float64, shape =
([layer_dims[0],None]))
    Y = tf.placeholder(dtype =
tf.float64, shape = ([1,None]))

    tf.set_random_seed(1)
    parameters = {}
    for i in
range(1,len(layer_dims)):
        parameters['W' + str(i)] =
tf.get_variable("W"+ str(i),
shape=[layer_dims[i], layer_dims[i-
1]],
initializer=tf.contrib.layers.xavier
_initializer(), dtype=tf.float64)
        parameters['b' + str(i)] =
tf.get_variable("b"+ str(i),
initializer=tf.zeros([layer_dims[i],
1],dtype=tf.float64))
```

```
    Z_final = forwardProp(X,
parameters, drop_out)

    cost =
tf.nn.sigmoid_cross_entropy_with_log
its(logits=Z_final,labels=Y)
    if optimizer == "momentum":
        train_net =
tf.train.MomentumOptimizer(0.01,
momentum=0.9).minimize(cost)
    elif optimizer == "rmsProp":
        train_net =
tf.train.RMSPropOptimizer(0.01,
decay=0.999, epsilon=1e-
10).minimize(cost)
    elif optimizer == "adam":
        train_net =
tf.train.AdamOptimizer(0.01, beta1 =
0.9, beta2 = 0.999).minimize(cost)
    if regularization:
        reg_term = 0
        L = len(parameters)//2
        for l in range(1,L+1):
```

```
            reg_term +=
tf.nn.l2_loss(parameters['W'+
str(l)])
        cost = cost + (lambd/2) *
reg_term
    cost = tf.reduce_mean(cost)

    train_net =
tf.train.GradientDescentOptimizer(0.
01).minimize(cost)
    init =
tf.global_variables_initializer()
    costs = []
    with tf.Session() as sess:
        sess.run(init)
        for i in range(10000):
            _,c =
sess.run([train_net, cost],
feed_dict={X: normalize(X_train).T,
Y: Y_train.reshape(1,
len(Y_train))})
            if i % 100 == 0:
                costs.append(c)
```

```
            if i % 1000 == 0:
                print(c)
        plt.ylim(min(costs)
+0.1 ,max(costs), 4, 0.01)
        plt.xlabel("epoches per
100")
        plt.ylabel("cost")
        plt.plot(costs)
        plt.show()
        params =
sess.run(parameters)
    return params

def predict(X, parameters):
    with tf.Session() as sess:
        Z = forwardProp(X,
parameters, drop_out= False)
        A =
sess.run(tf.round(tf.sigmoid(Z)))
    return A

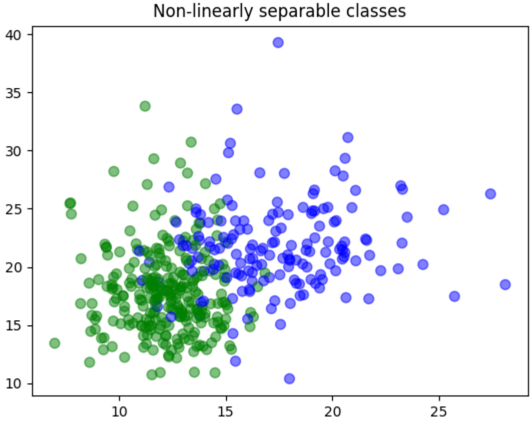
def plot_decision_boundary1( X, y,
model):
```

```
    plt.clf()
    x_min, x_max = X[0, :].min() -
1, X[0, :].max() + 1
    y_min, y_max = X[1, :].min() -
1, X[1, :].max() + 1
    colors=['blue','green']
    cmap =
matplotlib.colors.ListedColormap(col
ors)
    h = 0.01
    xx, yy =
np.meshgrid(np.arange(x_min, x_max,
h), np.arange(y_min, y_max, h))
    A = model(np.c_[xx.ravel(),
yy.ravel()])
    A = A.reshape(xx.shape)
    plt.contourf(xx, yy, A,
cmap="spring")
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.scatter(X[0, :], X[1, :],
c=y, s=8,cmap=cmap)
```

```
    plt.title("Decision Boundary for
learning rate:")
    plt.show()

p = deep_net(regularization = True,
lambd = 0.02)
plot_decision_boundary1(normalize(X_
train).T,Y_train,lambda x:
predict(x.T,p))

p = deep_net(drop_out = True)
p = deep_net(optimizer="momentum")
p = deep_net(optimizer="rmsProp")
p = deep_net(optimizer="adam")
```



0.8640964968487806

0.6576789363554733

0.5143526817019538

0.46109938219842395

0.4391590019944309

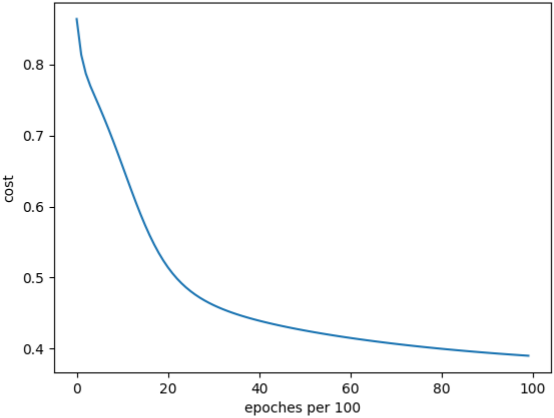
0.4253497158288082

0.41486545995180873

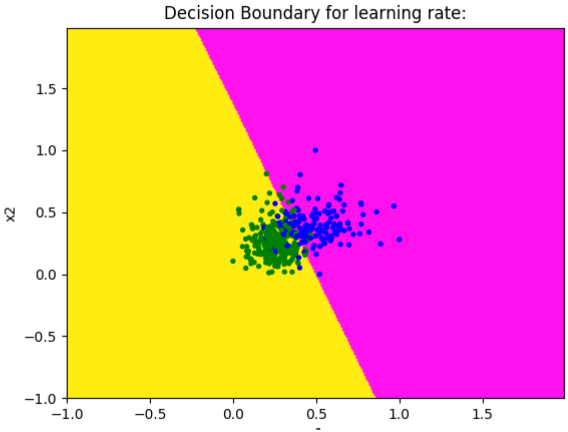
0.4064618870927915

0.39961302157076345

0.3940068025490444



0.8
0.7
0.6
0.5
0.4
cost
0
20
40
60
80
100
epoches per 100



0.7001094555968302

0.49777905860234667

0.36297676798298484

0.3099069894603214

0.29260054954117 37

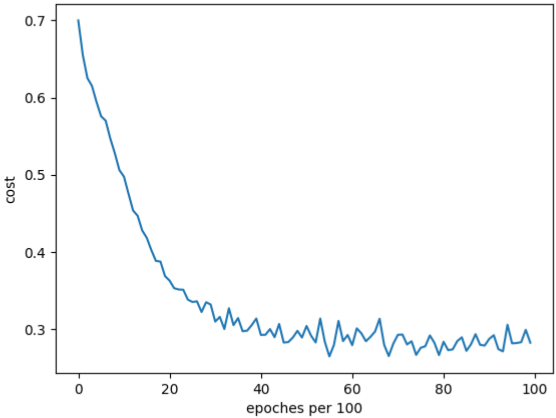
0.30422081040917726

0.279327552972575

0.2927437389848652

0.2837427041090086

0.2874399397021791



0.6979167571210456

0.46770023748893524

0.3114688059257045

0.271497538924154

0.26210240985355404

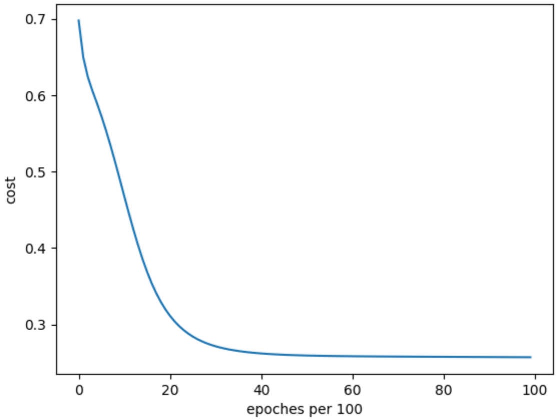
0.25939189901561954

0.25844236585729935

0.25798796421521397

0.25766904491870255

0.25738659596621496



0.6979167571210456

0.46770023748893524

0.3114688059257045

0.271497538924154

0.26210240985355404

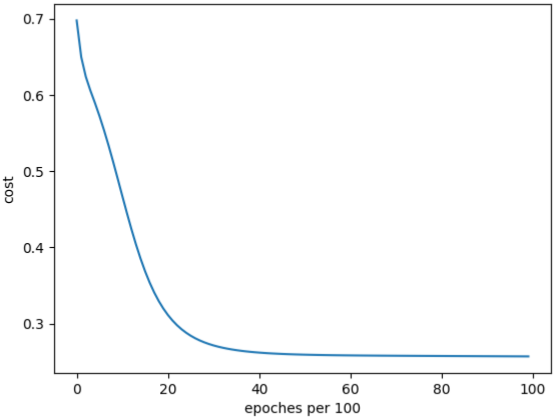
0.2593918901561954

0.25844236585729935

0.25798796421521397

0.2576690491870255

0.2573865959621496



0.6979167571210456

0.46770023748893524

0.3114688059257045

0.271497538924154

0.26210240985355404

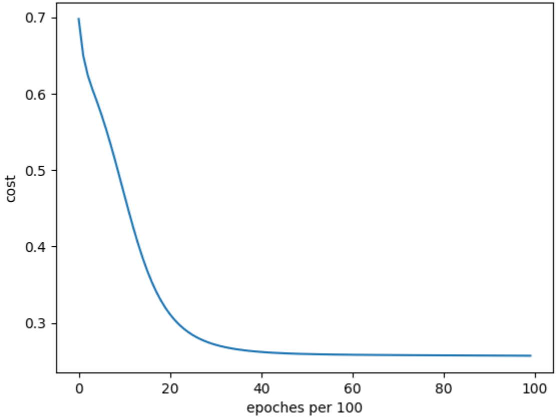
0.25939189015 61954

0.25844236585729935

0.25798796421521397

0.2576690491870255

0.2573865959621496


0.7
0.6
0.5
0.4
0.3
cost
0
20
40
60
80
100
epoches per 100

12 Convolutional Neural Networks

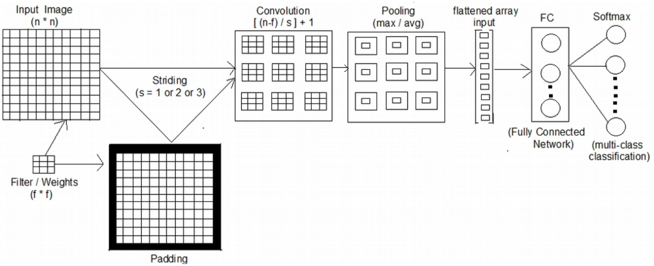
ஒரு database-ல் சேமிப்பதற்கு ஏற்ற வகையில் தரவுகளைக் கொண்ட அமைப்பிற்கு 'Structured data' என்று பெயர். இதுவரை நாம் பார்த்த அனைத்தும் ஒரு முறையான வடிவமைப்பைக் கொண்ட தரவுகளை நெட்வொர்குக்கு கொடுத்து எவ்வாறு பயிற்சி அளிப்பது என்று பார்த்தோம். இனிவரும் பகுதிகளில் ஒழுங்கற்ற தரவுகளுக்கான மாடல்களை எவ்வாறு உருவாக்குவது என்று பார்க்கலாம்.

அட்டவணை வடிவத்தில் அமையும் தரவுகள் 'structured data' என்றால், இவ்வடிவில் சேமிக்க இயலாத படங்கள், காணொளிகள் , உரை தரவுகள், குரல் கோப்புகள் (images, videos, text data, voice data) ஆகியவை 'unstructured data' ஆகும்.

CNN, RNN போன்றவை இத்தகைய வேலைகளுக்காகப் பயன்படுகின்றன. இவைகளைப் பற்றியெல்லாம் இனிவரும் பகுதிகளில் பார்க்கலாம்.

அதிக படங்களைக் கொண்டு பயிற்சி பெற்று புதிதாக வருகின்ற ஒரு படம் "என்ன படம்?" என கணிக்கும் வேலையை *CNN* சிறப்பாக செய்கிறது. பொதுவாக படங்களை வகைப்படுத்தி அடையாளம் *(Image classification & identification)* காணுவதற்கான ஒரு சிறப்பு வகை வடிவமைப்பில் இந்த *CNN* உருவாக்கப்பட்டுள்ளது. இவற்றில் பயன்படுத்தப்படும் சிறப்பு வகைப் பதங்களான *Convolution, Pooling, Striding, Padding, Flattened array, FC (fully connected network)* போன்றவை என்னென்ன வகைகளில் இந்த செயலுக்கு உதவுகின்றன? அவற்றையெல்லாம் எந்தெந்த இடங்களில் பயன்படுத்த வேண்டும்? போன்றவை பற்றியெல்லாம் இப்பகுதியில் காணலாம். இதன் கட்டமைப்பு பின்வருமாறு

இருக்கும். இதில் உள்ள ஒவ்வொரு படியையும் விளக்கமாகக் கீழே காணலாம்.



படி1 - Input Matrix:

முதலில் ஒரு படத்தை கணினி புரிந்து கொள்வதற்கு ஏற்ற வகையில் எவ்வாறு அமைக்க வேண்டும் என்பது நமக்குத் தெரிந்திருக்க வேண்டும். கணினியைப்

பொருத்த வரை ஒரு படம் என்பது பல்வேறு பிக்சல்களால் உருவானது. இந்தப் பல்வேறு நிறப் பிக்சல்களைக் கொண்ட படத்தை பல்வேறு எண்களைக் கொண்ட ஒரு அணியாக மாற்றி பின்னர்தான் CNN-க்கு உரிய பல்வேறு வேலைகளைத் துவங்க வேண்டும். RGB என்பது முதன்மை வண்ணங்கள். இவை ஒவ்வொன்றும் தன்னுடைய shades-ன் தீவிரத்தைப் பொறுத்து 0-255 வரை அமைந்த எண்களால் குறிக்கப்படும். இதுவே colour codes என்றழைக்கப்படுகிறது. இத்தகைய எண்களின் கலப்பே பல்வேறு நிறப் பிக்சல்களால் ஆன படத்தை ஒரு அணியாக மாற்ற உதவும். நியூரல் நெட்வொர்கைப் பொறுத்தவரை இத்தகைய ஒவ்வொரு பிக்சல் மதிப்பும் 'என்ன படம்?' என்பதைக் கணிக்க உதவும் features-ஆக அமையும். எனவே இத்தகைய features-ன் எண்ணிக்கையைக் குறைத்து நெட்வொர்குக்கு அனுப்ப CNN பல்வேறு வழிவகைகளைக் கையாள்கிறது. அவைகளைப் பற்றி இப்பகுதியில் காணலாம். முதலில் அமையும் அணி Input Matrix என்றழைக்கப்படுகிறது. இதற்கு அடுத்த

படியாக 'Convolution' என்ற ஒன்று நடைபெற்று features-ன் எண்ணிக்கையைக் குறைக்கிறது.

படி2 - Convolution:

இப்படியில் நம்முடைய படத்தின் பரிமாணம் குறைக்கப்படுகிறது. 'பரிமாணத்தைக் குறைத்தல்' என்றால் படத்தை எவ்விதத்திலும் வெட்டி எடுக்காமல் வெறும் பெரிய அளவு படத்தை சிறிய அளவில் மாற்றுவதைக் குறிக்கும். இதன் மூலம் படத்தின் தீவிரம் குறைந்தாலும், படத்திற்கு எவ்வித இழப்பும் ஏற்படாது. இது பின்வரும் முறையில் நிகழ்கிறது.

முதலில் 2*2 அல்லது 3*3 போன்ற சிறிய வடிவமைப்பைக் கொண்ட ஒரு அணி வரையறுக்கப்படுகிறது. இதுவே பெரிய

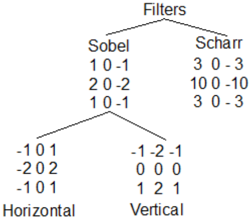
படத்தின் பரிமாணத்தைக் குறைக்க உதவும் ஒரு வடிகட்டி(filters) / அளவுறுக்கள்(parameters) ஆகும். இது நியூரல் நெட்வொர்கில் நாம் பயன்படுத்தும் weights-க்குச் சமமானது.

- *Scharr filter, Sobel filter போன்றவற்றை இத்தகைய வேலைகளுக்காகப் பயன்படுத்தலாம். Scharr filter என்பது சிறு சிறு பகுதிகளாகப் பிரிக்கப்படும் படத்தின் ஒவ்வொரு பகுதி முனைகளுக்கும் முக்கியத்துவம் கொடுத்து வடிகட்டி எடுக்கும். இதுவே 'Edge Detection' என்று அழைக்கப்படுகிறது.*

- *Sobel filter என்பது ஒரு படத்தின் மையப் பகுதியிலுள்ள பிக்சல்களுக்கு அதிக முக்கியத்துவம் கொடுத்து அதனடிப்படையில் வடிகட்டும். இத்தகைய வடிகட்டியின் மதிப்புகள் பின்வருமாறு.*

- மேலும் இந்த வடிகட்டும் நிகழ்வானது கிடைமட்டவாக்கில் நிகழ வேண்டுமா அல்லது செங்குத்துவாக்கில் நிகழ வேண்டுமா என்பதைப் பொறுத்து இவற்றில் அமையும் சுழியம் மதிப்புகள் இடம்மாறுகின்றன.

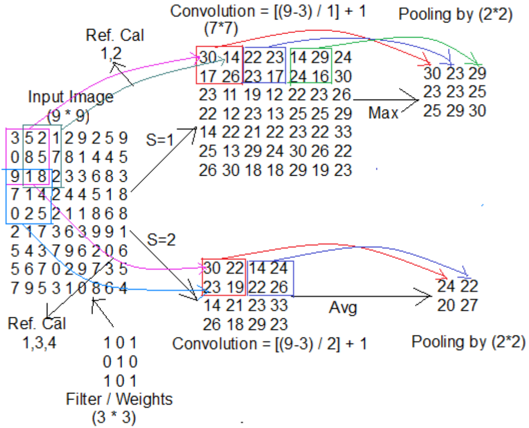
- இவைகளல்லாமல் நாமே கூட நமக்கேற்றவாறு மதிப்புகளைக் கொண்ட ஒரு வடிகட்டியை உருவாக்கிப் பயன்படுத்தலாம். இதுவே நம்முடைய மாடல் பயன்படுத்துகின்ற parameters ஆகும். எனவே back propagation முறைப்படி இது தனக்கான சரியான மதிப்பினைக் கற்றுக் கொண்டுவிடும்.



இதுபோன்ற ஒரு வடிகட்டியின் அளவில் பொருந்தும் படி நம்முடைய உள்ளீட்டு அணியானது முழுவதும் சிறு சிறு பகுதிகளாகப் பிரிக்கப்படுகிறது.

கீழ்க்கண்ட எடுத்துக்காட்டில் 3,5,2,1 .... எனத் துவங்கும் உள்ளீட்டு அணியும், 1,0,1 எனும் மதிப்பினைப் பெற்றுத் துவங்கும் 3*3 filter அணியும் வரையறுக்கப்பட்டுள்ளது. இது தன்னுடைய அளவில் பொருந்தும் படியாக

*உள்ளீட்டு அணியின் இடது மூலையில் தொடங்கி படம் முழுவதையும் சிறு சிறு பகுதிகளாகப் பிரிக்கப்படுகிறது. முதலில் படத்தின் இடது மூலையிலிருந்து 3,5,2 எனும் மதிப்பினைப் பெற்றுத் துவங்கும் 3*3 அணியை எடுக்கிறது.*



பின்னர் இச்சிறிய அணி மற்றும் வடிகட்டி அணி எனும் இவ்விரண்டு அணிகளின் பெருக்கல் நடைபெற்று புதிய அணியை உருவாக்குகிறது. இந்தப் புதிய அணியில் உள்ள மதிப்புகள் அனைத்தும் கூட்டப்பட்டு 30 எனும் சிறிய எண் வடிவில் குறிக்கப்படுகிறது. இதுவே "Convolved output" என்றழைக்கப்படும் அணியில் இடம்பெறும் முதல் மதிப்பாகும்.

```
       352   101     3*1 5*0 2*1     302
Cal-1 =085 * 010  =  0*0 8*1 5*0  =  080 = 3+2+8+9+8 = 30
30     918   101     9*1 1*0 8*1     908

       521   101     5*1 2*0 1*1     501
Cal-2 =857 * 010  =  8*0 5*1 7*0  =  050 = 5+1+5+1+2 = 14
14     182   101     1*1 8*0 2*1     102

       212   101     2*1 1*0 2*1     202
Cal-3 =578 * 010  =  5*0 7*1 8*0  =  070 = 2+2+7+8+3 = 22
22     823   101     8*1 2*0 3*1     803

       918   101     9*1 1*0 8*1     908
Cal-4 =714 * 010  =  7*0 1*1 4*0  =  010 = 9+8+1+0+5 = 23
23     025   101     0*1 2*0 5*1     005
```

இவ்வாறு "Convolved output" -ன் அடுத்தடுத்த மதிப்பினை உருவாக்கப் போகும் அணி முதல் அணியிலிருந்து எவ்வளவு இடைவெளி விட்டு தேர்ந்தெடுக்க வேண்டும் என்பதையே stride எனும் parameter குறிக்கிறது. மேற்கண்ட எடுத்துக்காட்டில் stride (S=1) எனும்போது முதல் அணியிலிருந்து ஒரு இடைவெளி விட்டு 5,2,1 எனும் மதிப்பைக் கொண்ட அடுத்த அணி தேர்ந்தெடுக்கப்படுகிறது. அதுவே stride=2

எனும்போது முதல் அணியிலிருந்து இரண்டு இடைவெளி விட்டு *2,1,2* எனும் மதிப்பினைக் கொண்ட அணி அடுத்த அணியாகத் தேர்ந்தெடுக்கப்படுவதைக் காணலாம். பின்னர் தேர்ந்தெடுக்கப்பட்ட இத்தகைய அணிகளுடன் பெருக்கல், கூட்டல் ஆகியவை நடைபெற்று *"Convolved output"* -ன் அடுத்தடுத்த மதிப்புகள் உருவாக்கப்படுகின்றன. இவ்வாறாக *stride* மதிப்பினைப் பொறுத்து *convolution* நடைபெறுகிறது. ஒரு *n*n* அணியை *f*f* வடிகட்டி மூலம் வடிகட்டும்போது உருவாகும் குறைந்த பரிமாணத்தைக் கொண்ட அணியானது *[(n-f) / s] + 1* எனும் வாய்ப்பாட்டின் படி அமையும். மேற்கண்ட எடுத்துக்காட்டில் *9*9* அணியை *3*3* வடிகட்டி மூலம் வடிகட்டும்போது கிடைக்கும் *convolved output*-ன் மதிப்பு,

- *stride=1* எனில் *7*7* எனவும் *i.e [(9-3) / 1] + 1,*

- *stride=2 எனில் 4*4 எனவும் i.e [(9-3) / 2] + 1, மேற்கண்ட படி அமைந்திருப்பதைக் காணலாம்.*

  இதற்கு அடுத்த படியாக 'Pooling' என்பதைப் பற்றிக் காணலாம்.

படி3 - Pooling:

ஒரு படத்தின் பரிமாணத்தைக் குறைப்பது 'convolution' என்றால், அப்படத்தின் அளவைக் குறைப்பது 'Pooling' எனப்படும். அதாவது குறைக்கப்பட்ட பரிமாணம் கொண்ட ஒரு படத்தின் மீது மீண்டும் ஒரு வடிகட்டியை செயல்படுத்தி, அதனை சிறு சிறு பகுதிகளாகப் பிரிக்கிறது. பின்னர் ஒவ்வொரு பகுதியிலிருந்தும் ஒருசில பிக்சல்களை மட்டும் தேர்ந்தெடுப்பதன் மூலம் அப்படத்தின் அளவு குறைக்கப்படுகிறது. இம்முறையில் பிக்சல்களைத் தேர்ந்தெடுக்க பல்வேறு வழிவகைகளை Pooling கையாள்கிறது. ஒரு வடிகட்டி மூலம் படமானது பல்வேறு

பகுதிகளாகப் பிரிக்கப்பட்ட பின்னர், ஒவ்வொரு பகுதியிலும் உள்ள பிக்சல்களில் பெரியதை தேர்வு செய்வது, சிறியதை தேர்வு செய்வது, அவைகளின் சராசரி மதிப்பினைத் தேர்வு செய்வது முறையே *Max pooling, Min pooling, Avg pooling* என்று அழைக்கப்படுகின்றன. மேற்கண்ட எடுத்துக்காட்டில் 2*2 வடிகட்டி அணி பயன்படுத்தப்பட்டுள்ளது. எனவேதான் 2*2 அமைப்பில் 'convolved output' -ஆனது சிறு சிறு பகுதிகளாகத் தேர்வு செய்யப்படுகிறது.

முதலில் 'convolved output' -ன் இடது மூலையிலிருந்து 30,14,17,26 எனும் பிக்சல் மதிப்புகளைக் கொண்ட சிறுபகுதி தேர்ந்தெடுக்கப்படுகிறது. பின்னர் இதில் 'Max Pooling' பயன்படுத்தப்பட்டு இவைகளில் பெரிய மதிப்பான 30 என்பது தேர்ந்தெடுக்கப்படுகிறது. இவ்வாறு அடுத்தடுத்த பகுதியிலிருந்தும் 'Max pooling' மூலம் மதிப்புகள்

தேர்ந்தெடுக்கப்படுவதால் படத்தின் அளவு குறைக்கப்படுகிறது.

விருப்ப படி - *Padding:*

இதை கட்டாயமாக்க வேண்டிய அவசியமில்லை. நீங்கள் விருப்பப்பட்டால் இதைப் பயன்படுத்திக் கொள்ளலாம். *Padding* என்பது படத்தின் நான்கு மூலைகளிலும் சுழியம் நிறைந்த வரிசைகளை உருவாக்கி படத்திற்கு எல்லை போன்ற ஒரு வடிவமைப்பைக் கொடுக்கும். இதுவே *Zero padding* என்று அழைக்கப்படுகிறது. ஏனெனில் மேற்கூறிய முறைகளில் அளவுகள் குறைக்கப்படும்போது, படத்தின் நான்கு மூலைகளிலும் உள்ள பிக்சல் மதிப்புகள் நீக்கப்படலாம். இதனால் அத்தகைய முனைகளிலிருந்து நமக்குக் கிடைக்க வேண்டிய சில முக்கியத் தகவல்கள் கிடைக்காமல் போவதற்கான வாய்ப்புகள் உள்ளன. இதனைத்

தவிர்ப்பதற்காக வந்ததே *Padding* ஆகும்.. இதன் மூலம் மூலைகளிலுள்ள பிக்சல் மதிப்புகள் நீக்கப்பட்டாலும், படத்தின் எல்லையிலுள்ள சுழிய மதிப்புகள் நீக்கப்படுமே தவிர படத்திற்கு எந்தவித இழப்பும் ஏற்படாது..

மேற்கண்ட எடுத்துக்காட்டில் *9*9* அணியை *3*3* வடிகட்டி மூலம் வடிகட்டும்போது, கிடைக்கும் *convolved output*-ன் மதிப்பு *7*7* எனப் பார்த்தோம். இதுவே படத்தின் நான்கு மூலைகளிலும் இரண்டு வரிசைகளில் சுழிய மதிப்புகளை அளித்தால் *9*9* அணியானது *11*11* என மாற்றப்பட்டு கிடைக்கும் *convolved output*-ன் மதிப்பும் *9*9 i.e [(11-3) / 1] + 1* என்றே அமையும். எனவே *zero padding* பயன்படுத்தும்போது, *9*9* அணியானது *convolution*-க்குப் பிறகும் *9*9* அணியை உருவாக்குவதால் இது *"same padding"* என்ற பெயரில் அழைக்கப்படுகிறது. இவைகள் எதையும் பயன்படுத்தாமல் சுழிய மதிப்புகள் எதுவும் வழங்காமல், ஒரு படத்தை அப்படியே

பயன்படுத்துவதே *"valid padding"* என்ற பெயரில் அழைக்கப்படுகிறது.

கடைசி படி:

மேற்கூறிய *convolution* மற்றும் *pooling* திரும்பத் திரும்ப நடைபெற்று ஒரு படத்தின் அளவு சுருக்கப்படுகிறது. பின்னர் சுருக்கப்பட்ட படத்திலுள்ள பிக்சல் மதிப்புகளே அப்படத்தை அடையாளம் காணுவதற்கான *features* -ஆக அமையும். இதுவே *'Flattened array of input values'* என்று அழைக்கப்படுகிறது. இம்மதிப்புகளே *FC* எனப்படும் *'Fully connected network'*-க்குள் செலுத்தப்பட்டு கடைசி லேயரில் என்ன படம் என்பது வகைப்படுத்தப்படுகிறது.

கீழ்க்கண்ட எடுத்துக்காட்டில் ஒரு படமானது முதலில் எவ்வாறு உள்ளது. *Padding* மூலம்

படத்திற்கு எல்லைகள் எவ்வாறு அமைக்கப்படுகிறது. *Convolution*-க்குப் பின்னர் படம் எவ்வாறு மாறுகிறது. கடைசியாக *Pooling* மூலம் படம் எவ்வாறு சுருங்குகிறது என்பது காட்டப்பட்டுள்ளது. மற்ற நெட்வொர்க் போன்று *CNN*-ஐ ஆதியிலிருந்து உருவாக்கிக் காட்டுவது என்பது ஒரு ஆராய்ச்சிக் கட்டுரையை நிகழ்த்திக் காட்டுவதற்கு சமமாகும். எனவேதான் இந்த அளவிலேயே எடுத்துக்காட்டை நிறுத்திக் கொண்டேன். பொதுவாக ஏற்கெனவே உருவாக்கப்பட்ட *CNN* மாடல்களை எடுத்து நமக்கேற்றவாறு அதற்கு மாற்றிப் பயிற்சி அளித்துப் பயன்படுத்திக் கொள்வதே சிறப்பாக அமையும்.

https://gist.github.com/nithyadurai87/50cb753e78bba477e3ef8c01ea338c63

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

```
def zero_pad(X, pad):
    X_padded = np.pad(array = X,
pad_width = ((0,0),(pad,pad),
(pad,pad),(0,0)), mode = 'constant',
constant_values = 0)
    return X_padded

def conv_single_step(X_slice, W, b):
    conv = np.multiply(X_slice, W)
    Z = np.sum(conv)
    Z = np.add(Z, b)
    return Z

def conv_forward(X, W, b, hparams):
  stride = hparams["stride"]
  pad = hparams["pad"]
  m, h_prev, w_prev, c_prev =
X.shape
  f, f, c_prev, n_c = W.shape

  n_h = int((h_prev - f +
2*pad)/stride) + 1
```

```
  n_w = int((w_prev - f +
2*pad)/stride) + 1

  Z = np.zeros((m, n_h, n_w, n_c))
  A_prev_pad = zero_pad(X, pad)
  for i in range(m):
    for h in range(n_h):
      for w in range(n_w):
        for c in range(n_c):
           w_start = w * stride
           w_end = w_start + f
           h_start = h * stride
           h_end = h_start + f
           Z[i,h,w,c] =
conv_single_step(A_prev_pad[i,
h_start:h_end, w_start:w_end, :],
W[:,:,:,c], b[:,:,:,c])
  return Z

def max_pool(input, hparams):
    m, h_prev, w_prev, c_prev =
input.shape
    f = hparams["f"]
```

```
    stride = hparams["stride"]
    h_out = int(((h_prev -
f)/stride) + 1)
    w_out = int(((w_prev -f)/stride)
+ 1)
    output = np.zeros((m, h_out,
w_out, c_prev))
    for i in range(m):
        for c in range(c_prev):
            for h in range(h_out):
                for w in
range(w_out):
                    w_start = w *
stride
                    w_end = w_start
+ f
                    h_start = h *
stride
                    h_end = h_start
+ f
                    output[i, h, w,
c] = np.max(input[i,h_start:h_end,
w_start:w_end, c])
```

```
    assert output.shape == (m,
h_out, w_out, c_prev)
    return output

img = mpimg.imread('./cake.JPG')
print (img.shape)
X = img.reshape(1,142,252,3)

fig = plt.figure(figsize=(15,10))

ax1 = fig.add_subplot(2,2,1)
print("Shape of Image: ", X.shape)
ax1.imshow(X[0,:,:,:])
ax1.title.set_text('Original Image')

ax2 = fig.add_subplot(2,2,2)
X = zero_pad(X, 10)
print("After padding: ", X.shape)
ax2.imshow(X[0,:,:,:], cmap =
"gray")
ax2.title.set_text('After padding')

ax3 = fig.add_subplot(2,2,3)
```

```
W = np.array([[-1,-1,-1],[-1,8,-1],
[-1,-1,-1]]).reshape((3,3,1,1))
b = np.zeros((1,1,1,1))
hparams = {"pad" : 0, "stride": 1}
X = conv_forward(X, W, b, hparams)
print("After convolution: ",
X.shape)
ax3.imshow(X[0,:,:,0],
cmap='gray',vmin=0, vmax=1)
ax3.title.set_text('After
convolution')

ax4 = fig.add_subplot(2,2,4)
hparams = {"stride" : 1, "f" : 2}
X = max_pool(X, hparams)
print("After pooling :", X.shape)
ax4.imshow(X[0,:,:,0], cmap =
"gray")
ax4.title.set_text('After pooling')

plt.show()
```

(142, 252, 3)

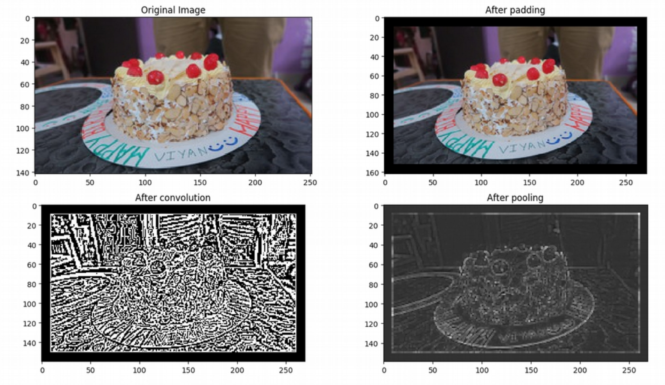
Shape of Image: (1, 142, 252, 3)

After padding: (1, 162, 272, 3)

After convolution: (1, 160, 270, 1)

After pooling : (1, 159, 269, 1)

13



ஒரு படத்தைக் கொடுத்து அதில் உள்ளது நாயா? பூனையா? என கணிக்கச் சொல்லுவது 'object classification' என்றால், ஒரு படத்தில் உள்ள ஒவ்வொரு பொருளும் என்னென்ன என கணிக்கச் சொல்லுவது 'object identification' ஆகும். இத்தகைய வேலைகளைச் செய்வதற்கென

ஏற்கெனவே பயிற்சி பெற்ற *AlexNet, ResNet, VGG19, InceptionResNet, GoogLeNet, DenseNet, NASNet* போன்ற மாடல்கள் சந்தையில் உள்ளன. இவற்றில் ஏதாவதொன்றைத் தேர்ந்தெடுத்து நமது தரவுகளுக்கேற்ப பயிற்சி அளித்து பயன்படுத்திக் கொள்ளலாம்.

ImageNet என்பது வலைத்தளத்திலிருந்து எடுக்கப்பட்ட பல்வேறு படங்களைக் கொண்ட ஒரு *database* ஆகும். இது ஒவ்வொரு வருடமும் *ILSVRC (ImageNet Large Scale Visual Recognition Challenge)* என்ற நிகழ்வினை நடத்துகிறது. இந்தச் சவாலில் மேற்கூறிய மாடல்கள் அனைத்தும் பங்கு பெறுகின்றன. இவைகளின் துல்லியத் தன்மையை உறுதிபடுத்த இதுபோன்ற சவால்கள் ஒவ்வொரு வருடமும் நிகழ்த்தப்படுகின்றன. எனவே இத்தகைய மாடல்களில் ஏதாவதொன்றை எடுத்து நமக்கு வேண்டிய படத்தைக் கொடுத்து கணிக்கச்

சொன்னால், அதன் கணிப்பு பொதுவாக சரியாகத்தான் இருக்கும்.

.1 Computer Vision

நமது கணினிக்கு ஒரு படத்தைப் பார்த்து அதிலுள்ள விஷயங்கள் என்னென்ன என அடையாளப்படுத்தக் கற்றுக் கொடுப்பதே *computer vision* என்று அழைக்கப்படும். இத்தகைய *object detection*-க்கான வேலையைச் செய்வதே *YOLO* எனும் *algorithm* ஆகும். *You Only Look Once* என்பதே *YOLO* என்று அழைக்கப்படுகிறது. இது ஒரு படத்தை எடுத்துக்கொண்டு, அதை சிறு சிறு துண்டுகளாக *grid*-வடிவில் பிரிக்கிறது. பின்னர் ஒவ்வொரு *grid*-ன் மீதும் *Image classification* மற்றும் *Localization* என்பது நடைபெறுகிறது. பின்னர் ஒவ்வொரு *object*-க்குமான *bounding boxes*-ஐ கணித்து, அது ஒவ்வொரு *class*-க்குள்ளும் அமைவதற்கான நிகழ்தகவையும் கணிக்கிறது. ஒரே *object* பல இடங்களில் மீண்டும் மீண்டும் கணிக்கப்படுவதைத் தவிர்ப்பதற்கு *bounding*

boxes-ஐ இன்னும் துல்லியமாக அமைக்க உதவுவதே *Intersection Over Union* மற்றும் *Non-max supression* போன்ற விஷயங்கள் ஆகும்.

Face Recognition என்பது தற்போது எங்கும் பரவலாகப் பயன்படுத்தப்பட்டு வருகின்ற ஒரு விஷயம் ஆகும். ஒவ்வொருவரின் கைப்பேசியிலும் கூட இதற்கான *app* உள்ளது. இந்த வேலையைச் செய்வதற்கு *YOLO* உதவுகிறது. இதனை *face verification* மற்றும் *face recognition* என இரண்டாகப் பிரித்துப் புரிந்து கொள்ளலாம். நமது கைபேசியில் உள்ளது போல ஒருவருடைய முகத்தைக் காண்பித்து அது இன்னாரது முகமா இல்லையா எனக் கூறுவது *face verification* ஆகும் *(binary classification – one to one mapping).*

Face recognition என்பது ஒருவருடைய முகத்தைக் கொடுத்து அதனை *db*-ல் உள்ள பல்வேறு முகங்களுடன் ஒப்பிட்டு எந்த முகத்துடன் பொருந்துகிறது எனக் கண்டுபிடிப்பது *face recognition* ஆகும். *ConvNet* என்பது இந்த வேலையைச் செய்தாலும், இது ஒவ்வொருவருடைய முகத்துக்குமான ஒரு மாதிரிப் படத்தை எடுத்துக்கொண்டு சேமிக்கிறது. இதுவே *'One short learning'* என்று அழைக்கப்படும். இம்முறையில் சேமிக்கும்போது, ஏற்கெனவே சேமிக்கப்பட்டுள்ள நபரின் முகமே மற்றொரு முறை வந்தால் கூட அதில் சற்று மாறுபட்ட *features* காணப்படின் அதனை மற்றொரு நபராக சேமிக்கிறது. எனவே "இவர் தான் அவர் " என சரியாக கணிக்கவும் தவறுகிறது. இதனைத் தவிர்ப்பதற்காக வந்ததே *Siamese Network* ஆகும்.

Siamese Network என்பது *similarity function*-ஐப் பயன்படுத்தி இத்தகைய வேலையைச்

செய்கிறது. இது உள்ளீடாக வருகின்ற படத்திற்கும் db-ல் சேமிக்கப்பட்டுள்ள படத்திற்குமான வேறுபாட்டை முதலில் கணிக்கிறது. பின்னர் இந்த வேறுபாடு எந்த அளவு வரை அமையலாம் என்பதற்கான threshold-ஐ அமைக்கிறது. இது வெளிப்படுத்துகின்ற வேறுபாடு threshold-க்கும் குறைவாக இருப்பின் "முகம் பொருந்துகிறது" எனவும், அதிகமாக இருப்பின் "முகம் பொருந்தவில்லை" எனவும் கணிக்கிறது. இத்தகைய முறையில் ஒருவருடைய முகம் கணிக்கப்படும்போது ஏற்படுகின்ற இழப்பைக் கண்டறிய உதவுவதே triplet loss function ஆகும். Anchor img என்பது கொடுக்கப்பட்டுள்ள படம் ஆகும். positive ,negative images என்பது கொடுக்கப்பட்டுள்ள படத்துடன் பொருந்துகின்ற மற்றும் பொருந்தாத படங்கள் ஆகும். இது anchor image, positive image, negative image என்று 3 படத்தை ஒப்பிட்டு இழப்பைக் கணக்கிடுகிறது. எனவேதான் triplet loss என்ற பெயரில் அழைக்கப்படுகிறது.

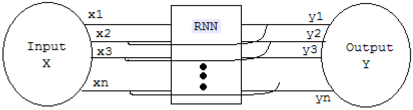
15 Recurrent Neural Networks

சாதாரண நியூரல் நெட்வொர்கில் அடுத்தடுத்து வரும் இரண்டு உள்ளீட்டுத் தரவுகள் ஒன்றோடொன்று தொடர்பில்லாமல் இருக்கும். எடுத்துக்காட்டாக ஒரு வீட்டினுடைய சதுர அடி விவரத்தைப் பெற்றுக்கொண்டு அதன் விலையைக் கணிக்கும் சோதனையை எடுத்துக்கொண்டால் 400 சதுர அடி வீட்டிற்கு ஒரு விலையையும், 600 சதுர அடி வீட்டிற்கு ஒரு விலையையும் கணிக்கும். இந்த இரண்டும் ஒன்றோடொன்று தொடர்பில்லாத விவரங்கள். 600 சதுர அடிக்கான விலையைக் கணிக்க ஒரு நெட்வொர்க் இதற்கு முந்தைய 400 சதுர அடி வீட்டிற்கு என்ன விலையைக் கணித்தோம் என்பதை நினைவில் வைத்துக் கொள்ளத் தேவையில்லை. இது போன்ற ஒன்றோடொன்று

தொடர்பில்லாத உள்ளீட்டுத் தரவுகளுக்கு சாதாரண நெட்வொர்கைப் பயன்படுத்தலாம்.

ஆனால் சில சமயங்களில் ஒரு விஷயத்தைக் கணிப்பதற்கு தொடர்ச்சியாக வருகின்ற உள்ளீட்டுத் தரவுகள் அனைத்தும் ஒன்றோடொன்று தொடர்புடையதாக அமைகின்றன. எடுத்துக்காட்டுக்கு பங்குச் சந்தைகளில் கணிக்கப்படும் ஒரு பங்கின் விலையானது அதன் தற்போதைய விலையைப் பொறுத்து மட்டும் அமையாமல் இதற்கு முன்னர் அமைந்த விலைகளின் பட்டியலைக் கணக்கில் கொண்டே கணிக்கப்படுகிறது. அது போலவே ஒரு சொற்றொடரைக் கேட்டு கணினி என்ன மாதிரியான பதிலை அளிக்க வேண்டும் என்பது வெறும் ஒரு சொல்லைப் பொறுத்து மட்டும் அமையாமல், பல்வேறு சொற்களின் தொடர்ச்சியைக் கொண்டே கணிக்கப்படுகிறது. இதுபோன்று அமையும் தரவுகளே *'sequential data'* என்று அழைக்கப்படுகிறது. இத்தகைய தரவுகள்

temporal மற்றும் *time-component* ஐப் பெற்று விளங்கும். இதுபோன்ற தரவுகளைப் பயன்படுத்தி கணிப்புகளை நிகழ்த்துவதற்கு ஏற்ற வகையில் *RNN* சிறப்பான வடிவமைப்பைப் பெற்று விளங்குகிறது.



இதன் வடிவமைப்பு ஒரே ஒரு நியூரானைக் கொண்டது. முதலில் வருகின்ற தரவின் அடிப்படையில் அந்த நியூரான் கொடுக்கும் வெளியீடு சேமிக்கப்பட்டு அடுத்து வருகின்ற தரவுடன் இணைக்கப்படுகிறது. பின்னர் இவையிரண்டின் அடிப்படையில் கிடைக்கும் வெளியீடு சேமிக்கப்பட்டு மீண்டும் அடுத்தடுத்து வருகின்ற தரவுகளுடன் தொடர்ச்சியாக இணைக்கப்படுகின்றன.

கடைசியாக இவைகளின் அடிப்படையிலேயே கணிப்புகள் நிகழ்த்தப்படுவதால் இது *Recurrent* என்ற பெயரில் அழைக்கப்படுகிறது. ஒவ்வொரு முறையும் கிடைக்கும் தற்காலிக வெளியீடுகளை சேமித்து வைக்க இது உள்ளூர ஒரு *memory*-ஐப் பயன்படுத்துகிறது. *RNTN* என்ற நெட்வொர்கில் இந்த *memory*-க்கு *LSTM (Long Short Term Memory)* என்று பெயர். இந்த இடத்தில்தான் என்னென்ன விஷயங்களெல்லாம் இதுவரை சேமிக்கப்பட்டுள்ளன எனும் விவரம் ஒவ்வொரு முறையும் கால-நேரத்தின் அடிப்படையில் சேமிக்கப்படுகிறது.

இதுபோன்று தொடர் நிகழ்வுகளை நினைவில் வைத்துக் கொண்டு புரிந்து செயல்படுவதற்கு ஏற்ற வகையான கட்டமைப்பைப் பெற்று விளங்குவதால் *'Text Processing'* போன்ற செயல்களில் இதைப் பயன்படுத்தலாம். அதாவது ஒரு கோப்பில் அமைந்திருக்கும்

கதைகளையோ, கட்டுரைகளையோ கொடுத்து RNN-க்குப் பயிற்சி அளித்தால், அது அவ்வார்த்தைகளின் தொடர்புப்படி பயிற்சியைப் பெற்றுக் கொள்கிறது. பின்னர் புதிதாக ஒரு வார்த்தையைக் கொடுத்து, அதைத் தொடர்ந்து அமையவிருக்கும் வார்த்தையை கணிக்கச் சொன்னால், தான் பெற்றுக் கொண்ட பயிற்சியின் படி கொடுக்கப்பட்ட வார்த்தையுடன், அதிக தொடர்புடைய வார்த்தைகள் என்னென்ன என்பதை கணித்துச் சொல்லும். எனவே ஒருவரின் பேச்சைப் புரிந்து கொள்வது, அதற்கேற்ப பதில் அளிப்பது, மொழிமாற்றம் செய்வது, பல்வேறு குரல்களை ஆராய்வது, உரையாடல்களை நிகழ்த்துவது போன்ற வேலைகளுக்கு RNN-ஐப் பயன்படுத்தலாம்.

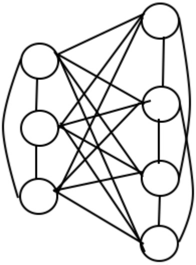
RNTN என்பது RNN-ன் ஒரு வகை. மேற்கூறிய அனைத்தையும் வெறும் ஆங்கில மொழிக்கு மட்டும் செய்யாமல், இன்ன பிற மொழிகளிலும்

செய்து காட்டுவதற்கு *NLP (Natural Language Processing)* என்று பெயர். அதாவது ஒரு மொழியிலுள்ள பெயர்ச்சொல், வினைச்சொல், உரிச்சொல், இணைப்புச்சொல் போன்றவற்றை அடையாளம் காணுதல் *(Parts of speech tagging)*, ஒரு சொல்லில் இருக்கும் வேர்ச் சொல்லை எடுத்தல் மற்றும் அதனைத் தழுவி வருகின்ற பிற சொற்களை அடையாளம் காணுதல் *(Stemming & Lemmatization)*, சொற்களின் பாகுபாடு*(segmentation)* போன்ற ஒரு மொழியைப் பற்றிப் புரிந்து கொள்ளத் தேவையான அனைத்து அம்சங்களையும் *RNTN* பெற்று விளங்குகிறது. *Multi-layer perceptron* என்பதையும் இந்த வேலைக்காகப் பயன்படுத்தலாம். ஆனால் இந்த *RNTN* இன்னும் சற்று மேம்பட்ட அம்சங்களுடன் இவ்வேலையைச் சிறப்பாகச் செய்கிறது.

16 BM, RBM, DBN Networks

Boltzmann Machines என்பதிலிருந்து உருவானதே *Restricted boltzmann machines* ஆகும். முதலில் *Boltzmann Network* என்றால் என்ன என்று பார்ப்போம். மாதிரித் தரவுகளில் உள்ள அதிகப்படியான *features*-ல் இருந்து நமக்கு வேண்டிய முக்கிய அம்சங்களை மட்டும் உருவாக்கும் வேலையை *Boltzmann Machines* செய்கிறது. இது வெறும் *input* மற்றும் *hidden* லேயரை மட்டும் பெற்று விளங்கும் நெட்வொர்க் ஆகும். மற்ற *deep learning* மாடலில் உள்ளது போன்று *output* லேயர் என்ற ஒன்று தனியாகக் கிடையாது. இந்த இரண்டு லேயரில் உள்ள *nodes* அனைத்தும் *undirected* முறையில் இணைக்கப்பட்டிருக்கும். அதாவது மற்ற நெட்வொர்க் போன்று தரவுகளை முன்னோக்கி

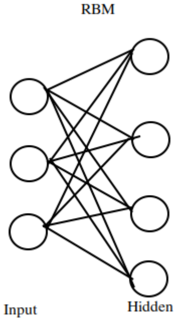
மட்டுமே செலுத்தாமல், இவற்றில் உருவாக்கப்படும் தரவுகள் அனைத்து திசையிலும் செலுத்தப்பட்டு மற்ற *nodes*-ஐ உருவாக்கும். எனவேதான் இது *'Generative Deep Learning Model'* என்று அழைக்கப்படுகிறது. இதன் அமைப்பு பின்வருமாறு.



இதில் உள்ள ஒவ்வொரு *nodes*-ம் சிறப்பு அம்சங்களைக் கண்டறிய உதவும் ஒரு *feature detector*-ஆக செயல்படும். இதில் இணைப்பு முறை எவ்வாறு இருக்குமென்றால், இரண்டு லேயரில் உள்ள *nodes*-ம் ஒன்றோடொன்று இணைப்பை ஏற்படுத்திக் கொள்வதுடன், ஒரே லேயரில் உள்ள *nodes*-ம் தங்களுக்குள் இணைப்பை ஏற்படுத்திக் கொள்ளும். இதனால் உருவாகக் கூடிய அதிக அளவு *feature detectors*-

ஆல், நெட்வொர்கின் வேகம் குறைந்து காணப்படும். இதனைத் தவிர்ப்பதற்காக

வந்ததே *Restricted Boltzmann Machines* ஆகும். இதன் அமைப்பு பின்வருமாறு.

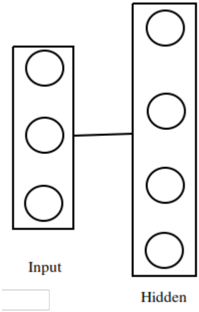


RBM என்பது ஒரே லேயரில் உள்ள *nodes* தங்களுக்குள் இணைப்பை ஏற்படுத்திக் கொள்வதை மட்டும் *restrict* செய்யும். மற்றபடி இதுவும் *BN*-ஐப் போன்றே இரண்டு லேயரைக் கொண்ட நியூரல் நெட்வொர்க் ஆகும். முதலாவது லேயர் *input* எனவும், இரண்டாவது லேயர் *hidden* எனவும் அழைக்கப்படும். இது ஒரே லேயரில் உள்ள *nodes*-க்குள் ஏற்படும் இணைப்பை மட்டும் *restrict* செய்வதால்,

'Restricted Boltzmann Network' என்ற பெயரில் அழைக்கப்படுகிறது. இதுவும் *undirected* என்பதால், பின்னோக்கிப் பரவுதல் மூலம் *gradient descent* முறையில் அளவுருக்களை சரி செய்வது என்பது இதில் கிடையாது. அதற்கு பதிலாக *'Contrastive divergence'* எனும் முறை பயன்படுத்தப்படுகிறது. இது ஆற்றலைக் குறைப்பதைக் குறிக்கோளாகக் கொண்டு செயல்படும் ஒரு இயற்பியல் சமன்பாட்டினை வைத்து செயல்படுகிறது. எனவே இது *'Energy based model'* வகையைச் சார்ந்தது எனலாம்.

dimensionality reduction, collaborative filtering, feature learning and topic modeling போன்ற இடங்களில் இது செயலாற்றி வருகிறது. *RBM* என்பது *visible nodes* வழியே வரும் உள்ளீட்டுத் தரவுகளுடன் *random* -ஆகத் தேர்ந்தெடுக்கப்பட்ட அளவுருக்களை*(weights)* இணைத்து *hidden nodes*-ஐ உருவாக்கும். பின்னர் உருவாக்கப்பட்ட *hidden nodes* அதே அளவுருக்களைப் பயன்படுத்தி அதே எண்ணிக்கையில் அமைந்த *visible nodes*-ஐ மறு உருவாக்கம் செய்கிறது.

Deep Belief Network என்பதை இன்னும் சற்று மேம்படுத்தப்பட்ட *RBM* எனலாம். இது *Stack of RBM* என்று அழைக்கப்படுகிறது. *RBM*-ன் ஒவ்வொரு லேயரிலும் உள்ள தனித்தனி *nodes*-ன் இணைப்புகளைத் தவிர்த்து, மாறாக லேயருக்கிடையேயான இணைப்பை ஊக்குவிக்கிறது. இது ஒன்றுக்கும் மேற்பட்ட *hidden* லேயர்களைக் கொண்டிருப்பின், ஒவ்வொரு லேயரும் தனக்கு முந்தைய லேயருடனும், தனக்கு அடுத்த லேயருடனும் இணைப்பை ஏற்படுத்திக் கொண்டு செயல்படுகிறது. தனக்கு முந்தைய லேயருக்கு *hidden* லேயராகவும், தனக்கு அடுத்த லேயருக்கு *visible* லேயராகவும் செயல்படுகிறது. ஒரே லேயருக்குள்ளோ அல்லது அருகாமையில் அமையாத லேயருடனோ இணைப்பை ஏற்படுத்தாது. இதன் அமைப்பு பின்வருமாறு.



மேற்கண்ட அனைத்தும் *UPN(Unsupervised Pretrained Network)* வகையைச் சார்ந்தது. அதாவது ஏற்கெனவே *unsupervised* முறையில் பயிற்சி அளித்து உருவாக்கப்பட்ட *algorithms* ஆகும். *RBM* என்பது *features*-ஐப் பிரித்தெடுத்து தரவுகளை மீண்டும் மறுஉருவாக்கம் செய்யப் பயன்படும். ஆனால் இதனால் கடினமான *features*-ஐக் கையாள முடியாது. படங்கள், ஒளி, ஒலி போன்றவற்றின் மீது செயல்பட்டு *cluster images, video capture sound, text* போன்ற வேலைகளைச் செய்வதற்கு *DBN* பெரிதும் பயன்படுகிறது.

கீழ்க்கண்ட எடுத்துக்காட்டில் 943 பயனர்கள், 1682 படங்களுக்கு வழங்கிய தரவரிசை மதிப்புகள் பயிற்சிக்கு, பரிசோதனைக்கு என இரண்டு கோப்புகளாக உள்ளன. 'u1.base' , 'u1.test' எனும் இரண்டு பெயரில் இருக்கும் இவற்றை கீழ்க்கண்ட முகவரியில் சென்று பதிவிறக்கம் செய்துகொள்ளலாம்.

https://www.kaggle.com/prajitdatta/movielens-100k-dataset

இத்தரவுகளில் உள்ள 1682 படங்களையும் தனித்தனி features-ஆக எடுத்துக்கொண்டு , நாம் RBM மூலம் 200 features-ஐ தேர்ந்தெடுக்கப்போகிறோம். எனவே RBM-ன் உள்ளீட்டு லேயரில் 1682 nodes-ம், hidden லேயரில் 200 nodes-ம் அமையும். இந்த உள்ளீட்டு லேயரில் உள்ள ஒவ்வொரு node-ம் ஒவ்வொரு படத்தைக் குறிக்கும். இதன் வழியேதான் மொத்தம் உள்ள 943 பயனர்களும் அக்குறிப்பிட்ட படத்திற்கு

அளித்த தரவரிசை மதிப்புகள் செலுத்தப்படும். பின்னர் sampling மூலம் 200 features தேர்ந்தெடுக்கப்படும். இதற்கான நிரல் பின்வருமாறு.

https://gist.github.com/nithyadurai87/74b40cab9cf65929c6fe1c99c1b58a9c

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from torch.autograd import Variable

def convert(data):
```

```
    new_data = []
    for i in range(1, 944):
        id_movies = data[:,1]
[data[:,0] == i]
        id_ratings = data[:,2]
[data[:,0] == i]
        ratings = np.zeros(1682)
        ratings[id_movies - 1] =
id_ratings

new_data.append(list(ratings))
    return new_data

W = torch.randn(200, 1682)
a = torch.randn(1, 200)
b = torch.randn(1, 1682)


def hidden(x):
    wx = torch.mm(x,W.t())
    activation = wx +
a.expand_as(wx)
    ph = torch.sigmoid(activation)
```

```
    return ph, torch.bernoulli(ph)

def visible(y):
    wy = torch.mm(y,W)
    activation = wy +
b.expand_as(wy)
    pv = torch.sigmoid(activation)
    return pv, torch.bernoulli(pv)

def train(v0, vk, ph0, phk):
    global W,a,b
    W += (torch.mm(v0.t(), ph0) -
torch.mm(vk.t(), phk)).t()
    b += torch.sum((v0 - vk), 0)
    a += torch.sum((ph0 - phk), 0)

training_set =
pd.read_csv('./u1.base', delimiter =
'\t')
test_set = pd.read_csv('./u1.test',
delimiter = '\t')
```

```
training_set =
np.array(training_set, dtype =
'int')
test_set = np.array(test_set, dtype
= 'int')

print (max(max(training_set[:,0]),
max(test_set[:,0])))
print (max(max(training_set[:,1]),
max(test_set[:,1])))

training_set = convert(training_set)
test_set = convert(test_set)

training_set =
torch.FloatTensor(training_set)
test_set =
torch.FloatTensor(test_set)

training_set[training_set == 0] = -1
training_set[training_set == 1] = 0
training_set[training_set == 2] = 0
training_set[training_set >= 3] = 1
```

```
test_set[test_set == 0] = -1
test_set[test_set == 1] = 0
test_set[test_set == 2] = 0
test_set[test_set >= 3] = 1

for epoch in range(1, 11):
    train_loss = 0
    s = 0.
    for i in range(0, 943, 100):
        vk = training_set[i:i+100]
        v0 = training_set[i:i+100]
        ph0,_ = hidden(v0)
        for k in range(10):
            _,hk = hidden(vk)
            _,vk = visible(hk)
            vk[v0<0] = v0[v0<0]
        phk,_ = hidden(vk)
        train(v0, vk, ph0, phk)
        train_loss += 
torch.mean(torch.abs(v0[v0>=0] - 
vk[v0>=0]))
        s += 1.
```

```
    print('epoch: '+str(epoch)+'
loss: '+str(train_loss/s))

test_loss = 0
s = 0.
for i in range(943):
    v = training_set[i:i+1]
    vt = test_set[i:i+1]
    if len(vt[vt>=0]) > 0:
        _,h = hidden(v)
        _,v = visible(h)
        test_loss +=
torch.mean(torch.abs(vt[vt>=0] -
v[vt>=0]))
        s += 1.
print('test loss:
'+str(test_loss/s))
```

நிரலுக்கான விளக்கம்:

1. முதலில் மாதிரித் தரவுகள் training_set, test_set எனும் பெயரில் 2 dataframes-ஆக சேமிக்கப்பட்டு பின்னர் numpy array வடிவில் மாற்றப்படுகின்றன.

```
training_set = pd.read_csv('./u1.base', delimiter = '\t')

test_set = pd.read_csv('./u1.test', delimiter = '\t')

training_set = np.array(training_set, dtype = 'int')

test_set = np.array(test_set, dtype = 'int')
```

இதில் உள்ள மதிப்புகள் பின்வருமாறு அமையும்.

```
print (training_set[:3])

[[        1         2         3   876893171]

 [        1         3         4   878542960]
```

[1 4 3 876893119]

[1 5 3 889751712]]

- ➔ *முதல் column-ல் உள்ளது 1 முதல் 943 வரை அமைந்த பயனருக்கான எண்ணைக் குறிக்கிறது.*

- ➔ *2 வது column-ல் உள்ளது 1 முதல் 1682 வரை அமைந்த படங்களுக்கான எண் ஆகும். அதாவது முதல் பயனர் எந்தெந்த படங்களுக்கு தரவரிசை மதிப்புகளை அளித்துள்ளார் என வெளிப்படுத்தியபின், 2-வது 3-வது என ஒவ்வொரு பயனரும் rating அளித்துள்ள படங்களை வெளிப்படுத்தும்.*

- ➔ *3 வது column-ல் ஒரு பயனர் அக்குறிப்பிட்ட படத்துக்குக் கொடுத்துள்ள தரவரிசை எண் மதிப்பு அமைந்திருக்கும். 1 முதல் 5 வரை*

அமைந்த எண்களால் ஒரு படத்தின் தரம் குறிக்கப்படுகிறது.

- 4 வது column எந்த நேரத்தில் அப்பயனர் தர மதிப்பைக் கொடுத்துள்ளார் என்பதை வெளிப்படுத்துகிறது.

மொத்தம் எத்தனை பயனர்களும், படங்களும் உள்ளன எனும் விவரம் பின்வருமாறு கண்டுபிடிக்கப்பட்டுள்ளது. ஏனெனில் இத்தரவுகள் *training, testing* என இரண்டாகப் பிரிந்திருப்பதால், இரண்டிலும் உள்ள பெரிய மதிப்பு கண்டுபிடிக்கப்பட்டு, அதில் பெரியது கண்டுபிடிக்கப்படுகிறது.

print (max(max(training_set[:,0]), max(test_set[:,0]))) - 943 Users

print (max(max(training_set[:,1]), max(test_set[:,1]))) - 1682 Movies

2. இப்போது RBM தனது வேலையைத் தொடங்குவதற்கு ஏற்றவாறு 943 rows மற்றும் 1682 columns-ல் அமைந்த தர வரிசை மதிப்புகளைக் கொண்ட ஒரு அணி உருவாக்கப்பட வேண்டும். இத்தகைய அமைப்பினை உருவாக்கும் வேலையைத்தான் convert() function செய்கிறது.

```
def convert(data):

   new_data = []

   for i in range(1, 944):

       id_movies = data[:,1][data[:,0] == i]

       id_ratings = data[:,2][data[:,0] == i]

       ratings = np.zeros(1682)

       ratings[id_movies - 1] = id_ratings
```

```
        new_data.append(list(ratings))

    return new_data
```

இந்த *function*-க்குள் முதலில் ஒரு காலிப் பட்டியல் உருவாக்கப்படுகிறது. பின்னர் *for loop* மூலம் ஒவ்வொரு பயனரும் எந்தெந்த படங்களுக்கு *rating* அளித்துள்ளார், என்னென்ன *rating* அளித்துள்ளார் எனும் விவரம் *id_movies* , *id_ratings* எனும் இரண்டு பட்டியல்களாக சேமிக்கப்படுகின்றன. இது பின்வருமாறு.

```
print (id_movies)

[  2   3   4   5   7   8   9  11  13  15  16  18  19  21  22  25
 26  28

  29  30  32  34  35  37  38  40  41  42  43  45  46  48  50
 52  55  57

  58  59  63  66  68  71  75  77  79  83  87  88  89  93  94
 95  99 101
```

105 106 109 110 111 115 116 119 122 123 124 126 127 131 133 135 136 137

138 139 141 142 144 146 147 149 152 153 156 158 162 165 166 167 168 169

172 173 176 178 179 181 182 187 191 192 194 195 197 198 199 203 204 205

207 211 216 217 220 223 231 234 237 238 239 240 244 245 246 247 249 251

256 257 261 263 268 269 270 271] = 134

print (id_ratings)

[3 4 3 3 4 1 5 2 5 5 5 4 5 1 4 4 3 4 1 3 5 2 1 2 3 3 2 5 4 5 4 5 5 4 5 5 4

5 2 4 4 3 4 4 4 3 5 4 5 5 2 4 3 2 2 4 5 1 5 5 3 5 3 4 5 2 5 1 4 4 3 5 1 3

3 2 4 4 3 2 5 3 4 3 4 5 5 2 5 5 5 5 5 5 3 5 4 4 5 4 4 5 5 5 4 4 5 3 5 3 5

3 3 5 1 4 2 4 4 3 2 2 5 1 4 4 4 4 1 1 5 5 5 2] = 134

முதல் பயனர் *134* படங்களுக்கு *rating* கொடுத்துள்ளார். எனவே முதல் பயனருக்கான இவ்விரண்டு பட்டியல்களும் மேற்கண்டவாறு காணப்படும். மொத்தமாக உள்ள *1682* படங்களில் *134* படங்களுக்கே *rating* கொடுத்துள்ளார். எனவே *rating* அளிக்காத படங்களுக்கு *0* எனும் மதிப்பினை நாம் வழங்க வேண்டும். எனவே அடுத்ததாக *0* -ஐ மட்டும் பெற்று விளங்கும் *ratings* எனும் பெயர் கொண்ட *1d array* உருவாக்கப்படுகிறது. இது பின்வருமாறு.

ratings = np.zeros(1682)

print (ratings)

[0. 0. 0. ... 0. 0. 0.]

இது *1682 columns*-ல் அமைந்த *0's* ஐப் பெற்றிருக்கும். பின்னர் எந்த படங்களுக்கு மட்டும் *rating* வழங்கப்பட்டுள்ளதோ, அந்த இடத்தில் மட்டும் *0*-ஐ நீக்கி வழங்கப்பட்ட *rating*-ஆல் மாற்றம் செய்யும். இதற்காக *ratings[id_movies - 1] = id_ratings* எனக் கொடுக்கப்பட்டுள்ளது. இந்த இடத்தில் *id_movies* எனக் கொடுக்காமல் *id_movies - 1* என கொடுக்கப்பட்டிருப்பதைக் கவனிக்கவும். ஏனெனில் *2* வது படத்துக்கு *3* எனும் மதிப்பீடு அளிக்கப்பட்டிருந்தால், இந்த *array*-ல் இரண்டாவதாக அமைந்திருக்கும் *0*-ஐ மாற்றுவதற்கு பதிலாக, மூன்றாவதாக அமைந்திருக்கும் *0*-ஐ மாற்றிவிடும். ஏனெனில் படங்களின் *index 1*-லிருந்து ஆரம்பிக்கிறது. ஆனால் *array*-ன் *index 0*-லிருந்து ஆரம்பிக்கிறது.

இதனைத் தவிர்ப்பதற்காகவே *id_movies - 1* என கொடுக்கப்பட்டுள்ளது.

இதே முறையில் மொத்தம் *943* பயனர்களுக்கு *943 1d array* உருவாக்கப்படுகிறது. இவை *new_data* எனும் பெரிய பட்டியலுக்குள் ஒவ்வொன்றாகச் சேர்க்கப்படுகின்றன. இப்போது *print(new_data[0])* எனக் கொடுத்துப் பார்த்தால், முதல் பயனர் *rating* அளித்துள்ள *2,3,4,5,7* ஆகிய இடங்களில் மட்டும் அவைகளுக்கான *rating* வெளிப்படுவதைக் காணலாம். *1,6* மற்றும் பல *rating* அளிக்காத இடங்களில் *0* வெளிப்படுவதைக் காணலாம். இது பின்வருமாறு.

[0.0, 3.0, 4.0, 3.0, 3.0, 0.0, 4.0, 1.0, 5.0, 0.0, 2.0, 0.0, 5.0, ... 0.0, 0.0]

3. மேற்கண்ட முறையில் அனைத்து பயிற்சித் தரவுகளையும் மாற்றி அமைத்து பின்னர் PyTorch ஏற்றுக் கொள்ளக் கூடிய array வடிவில் மாற்றுவதற்கான நிரல் பின்வருமாறு.

```
training_set = convert(training_set)

test_set = convert(test_set)

training_set = torch.FloatTensor(training_set)

test_set = torch.FloatTensor(test_set)
```

4. இது binary classification-க்கான problem என்பதால் தரவுகள் அனைத்தையும் 0's & 1's வடிவில் மாற்றப்போகிறோம். 1,2 எனும் குறைந்த மதிப்பீடுகள் 0-ஆலும், 3,4,5 எனும் அதிக அளவில் அமைந்த மதிப்பீடுகள் 1- ஆலும் இடமாற்றம் செய்யப்படுகின்றன. ஏற்கெனவே தரவரிசை அளிக்கப்படாததைக் குறிக்கும் 0

மதிப்பு *-1* ஆல் இடமாற்றம் செய்யப்படுகிறது. இதற்கான நிரல் பின்வருமாறு.

```
training_set[training_set == 0] = -1

training_set[training_set == 1] = 0

training_set[training_set == 2] = 0

training_set[training_set >= 3] = 1

test_set[test_set == 0] = -1

test_set[test_set == 1] = 0

test_set[test_set == 2] = 0

test_set[test_set >= 3] = 1
```

5. இப்போது பயிற்சிக்குச் செலுத்தப்படும் தரவுகள் பின்வரும் வடிவில் அமைந்திருக்கும். இதில் 943 rows மற்றும் 1682 columns காணப்படும்.

```
print (training_set)

print (training_set.size())

tensor([[-1.,  1.,  1.,  ..., -1., -1., -1.],

        [ 1., -1., -1.,  ..., -1., -1., -1.],

        [-1., -1., -1.,  ..., -1., -1., -1.],

        ...,

        [ 1., -1., -1.,  ..., -1., -1., -1.],

        [-1., -1., -1.,  ..., -1., -1., -1.],

        [-1.,  1., -1.,  ..., -1., -1., -1.]])
```

torch.Size([943, 1682])

பின்னர் *for loop* மூலம் *10* சுழற்சிகள் உருவாக்கப்பட்டு, ஒவ்வொரு சுழற்சியிலும் பயிற்சிக்குப் பயன்படுத்திய அளவுருக்களை சரி செய்யும் நிகழ்வு நடைபெறுகிறது. மேலும் மீண்டும் ஒரு *for loop* மூலம் நூறு நூறாக அமைந்த *10* தொகுதிகளில் தரவுகளைப் பிரித்து பயிற்சி அளிக்கிறது. *weights, bias* ஆகியவை பயிற்சிக்குப் பயன்படுத்தப்பட்ட அளவுருக்கள் ஆகும். இவை *random*-ஆன மதிப்புகளைப் பெற்று இப்பயிற்சியைத் தொடங்குகிறது. மேலும் ஒவ்வொரு சுழற்சியிலும் இத்தகைய அளவுருக்களை மேம்படுத்துவதற்கு *train()* எனும் *function*-ஐப் பயன்படுத்துகிறது. *Input* மற்றும் *hidden* லேயர்கள் இரண்டும் ஒரே *weights* மதிப்பினைப் பயன்படுத்துகின்றன. ஆனால் இதற்கான *bias* மதிப்புகள் மட்டும் வேறுபடுகின்றன. *a* என்பது *hidden* லேயருக்கானது . *b* என்பது *input* லேயருக்கானது. *weights* என்பது *random*-ஆகத்

தேர்ந்தெடுக்கப்பட்ட *(200,1682)* வடிவில் அமைந்த மதிப்புகளைப் பெற்றுத் துவங்கும்.

hidden() மற்றும் *visible() functions* இரண்டும் ஒவ்வொரு முறையும் *hidden node*-க்கான மாதிரிகளையும், *visible node*-க்கான மாதிரிகளையும் தேர்ந்தெடுக்கின்றன. இவைகள் இரண்டும் *2* மதிப்புகளைத் திருப்பி அளிக்கின்றன. முதலாவது மதிப்பு ஒவ்வொரு மாதிரியும் *sample*-ஆக அமைவதற்கான நிகழ்தகவைக் குறிக்கும். இரண்டாவது மதிப்பு *Bernoulli* முறையில் தேர்ந்தெடுக்கப்பட்ட *sample*-ஐப் பெற்றிருக்கும். மேலும் *hidden ()* வெளிப்படுத்துகின்றன தரவுகளின் வடிவம் *torch.Size([100, 200])* என இருப்பதையும், *visible ()* வெளிப்படுத்துகின்றன தரவுகளின் வடிவம் *torch.Size([100, 1682])* என இருப்பதையும் காணலாம்.

```
print (ph0)
```

tensor([[9.9967e-01, 2.0279e-21, 1.0000e+00, ..., 3.9762e-26, 1.0000e+00, 1.0000e+00],

[1.1460e-09, 2.4450e-22, 1.0000e+00, ..., 3.7041e-22, 1.0000e+00, 2.0576e-08],

[1.9920e-03, 1.8334e-11, 1.0000e+00, ..., 3.2599e-21, 1.0000e+00, 1.1196e-17],

...,

[3.3063e-03, 4.9096e-21, 1.0000e+00, ..., 1.1215e-25, 1.0000e+00, 2.0377e-05],

[9.8176e-01, 5.9265e-14, 1.0000e+00, ..., 6.0945e-25, 1.0000e+00, 2.3354e-03],

[2.9210e-09, 1.6359e-11, 1.0000e+00, ..., 2.2869e-15, 1.0000e+00, 2.2399e-13]])

இம்முறையிலேயே RBM, features-ஐத் தேர்வு செய்கிறது. பயிற்சி அளித்து முடித்தபின், அதே தரவுகளைப் பயன்படுத்தி hidden நியூரான்களைத் தூண்டுவதன் மூலம் இதனை நாம் சோதித்துக் கொள்ளலாம்.

17 Autoencoders

Autoencoder என்றால் தரவுகளைத் தானாகவே ஏதோ ஒரு முறையில் குறியிட்டு சுருக்கி அமைக்கக் கற்றுக் கொள்ளுகின்ற ஒரு விஷயம் என்று பொருள். எனவேதான் *dimensionality reduction, feature representation* போன்ற இடங்களில் இது பெரும்பங்கு வகிக்கிறது. *Machine learning*-க்கான அறிமுகக் கற்றலில் *PCA*-ஐப் பற்றிப் பார்த்தோம் அல்லவா, அதே வேலையைச் செய்வதற்காக நியூரல் நெட்வொர்கில் பயன்படுத்தப்படும் ஒரு விஷயமே *autoencoder* ஆகும். *PCA* என்பது நேர்கோடு முறையில் அமையும் தரவுகளின் *dimension*-ஐக் குறைக்கப் பயன்படுகிறதெனில் , *Autoencoder* என்பது நேர்கோட்டில் அமையாத(*non-linear*) தரவுகளையும் கையாள்கிறது. இதுவும் *unsupervised learning* வகையைச் சார்ந்தது. ஏனெனில்

இம்முறையில்தான் குறியிட்டு அமைக்க வேண்டும் என்பதற்கு எந்தஒரு பயிற்சியும், வழிகாட்டுதலும் இன்றி இது கற்றுக்கொள்கிறது.

இதன் முதலாவது லேயர் உள்ளீட்டு லேயர் என்று அழைக்கப்படும். அடுத்தடுத்து உள்ளது *encoding* மற்றும் *decoding* லேயர்கள் என்று அழைக்கப்படும். இதன் வேலையே உள்ளீட்டுத் தரவுகளை ஏதோ ஒரு முறையில் குறியிட்டு சுருக்கி அளிப்பது மற்றும் அக்குறியீட்டினை நீக்கம் செய்து மீண்டும் பழைய நிலையில் தரவுகளை வெளிப்படுத்துவது ஆகும். கடைசியாக உள்ளது வெளியீட்டு லேயர். இது வெளிப்படுத்துகின்ற மதிப்பும், உள்ளீட்டு லேயர் வழியே செலுத்தப்படுகின்ற மதிப்பும் ஒன்றாகவே அமையும். இடையில் *encoding* மற்றும் *decoding* செய்யப் பயன்படுகின்ற ஒரு விஷயமே *'bottleneck'* என்றும் அழைக்கப்படுகிறது. இது பொதுவாகப் படங்களின் மீது செயல்பட்டு தேவையில்லாத

இடங்களை அகற்றுவது *(denoising image)* , பரிமாணங்களைக் குறைப்பது *(dimensionality reduction)*, கருப்பு வெள்ளைப் படங்களை வண்ணப் படங்களாக மாற்றுவது *(Image colouring)* , படத்தின் முக்கிய அம்சங்களை மட்டும் தேர்ந்தெடுத்து வெளிப்படுத்துவது *(feature variation)* , சில படங்களின் மீது இயல்பாக மங்கிய நிலையில் அச்சிடப்பட்டுத் தென்படுகின்ற கடையின் பெயர், பதிப்புரிமை போன்ற எழுத்துக்களை நீக்குவது *(watermark removal)*போன்ற விஷயங்களைச் செய்வதற்கு இது பெரிதும் பயன்படுகிறது.

- *Denoising Autoencoder* என்பது ஒரு அடிப்படை வகை. இது சிதைந்த நிலையில் இருக்கும் தரவுகளையோ / படத்தையோ எடுத்துக் கொண்டு அவற்றிலிருக்கும் வலுவான அம்சங்களை *(robust features)* மட்டும் வைத்து மீண்டும் அத்தரவுகளையோ ,

படத்தையோ மறுஉருவாக்கம் செய்யப் பயன்படுகிறது.

- *Sparse Autoencoder* என்பது இடைப்பட்ட லேயரில் உள்ள *nodes*-ன் எண்ணிக்கையைக் குறைக்காமல், உள்ளீட்டு மற்றும் வெளியீட்டு லேயரில் உள்ள *nodes*-ன் எண்ணிக்கைக்குச் சமமாகவே அமைக்கிறது. இவ்வாறு செய்யும்போது தரவுகளை *encode* செய்யாமல் ஒவ்வொரு *node*-ம் அப்படியே நினைவில் ஏற்றிக் கொள்ளும் அபாயம் ஏற்படும். இதனைத் தவிர்ப்பதற்காக ஒவ்வொரு லேயரிலும் ஒருசில *nodes*-ன் செயல்பாட்டை முடக்கி வைப்பதன் மூலம் *Information bottleneck*-ஐ செயல்பட வைக்கிறது.

- *Deep Autoencoder* என்பது முதலாம் நிலை, இரண்டாம் நிலை, மூன்றாம் நிலை என பல்வேறு நிலைகளில் அடுக்கடுக்காக *features*-ஐக் குறைத்துக் கொண்டே வந்து

கடைசியில் அவற்றைக் குறியிட்டு வைக்கிறது. பின்னர் இக்குறியீட்டினை நீக்கம் செய்து, அத்தரவுகளை வெளிக்கொண்டு வருவதும் இதே முறையில் பல்வேறு நிலைகளில் நடைபெறுகிறது. *Image Search, Data compression, Topic Modeling, Information retrieval* போன்ற விஷயங்களில் இது பெரும்பங்கு வகிக்கிறது.

- *Contractive autoencoder* என்பது பெயரிடப்படாத தரவுகளை *(unlabeled data)* ஒரு குறிப்பிட்ட *label*-ன் கீழ் சுருக்கி வெளிப்படுத்த உதவுகிறது. இதுபோன்று உருவாக்கப்படும் லேபில் *latent variables* என்று அழைக்கப்படுகின்றன. இது *Regularized autoencoder*-ன் ஒரு வகை ஆகும்.

- *Variational autoencoder* என்பது ஒரே படத்தில் சிறுசிறு மாற்றங்களைச் செய்து புதுப்புது படங்களை உருவாக்குவதற்கு

உதவுகிறது. நாம் ஏற்கெனவே இதுபோன்ற காணொளிகளைப் பார்த்திருப்போம் அல்லவா! முதலில் ஒருவருடைய முகத்தில் ஆரம்பித்து அதிலிருந்து சிறுசிறு மாற்றங்களைச் செய்து, பலருடைய முகங்கள் தென்படுமே! இவையெல்லாம் *Variational autoencoder*-ன் வேலையே. *Graphical models* உருவாக்கம் மற்றும் *image generation* போன்ற இடங்களில் இது பெரும்பங்கு வகிக்கிறது.

18 Reinforcement Learning

கணினிக்கோ அல்லது கணினியால் உருவாக்கப்பட்ட கருவிக்கோ ஒரு விஷயத்தை திரும்பத் திரும்பச் சொல்லிக் கொடுப்பதன் மூலம் அதனைப் பயிற்றுவிக்க முயலும் முறைக்கு 'Reinforcement Learning' என்று பெயர். சுயமாக ஓடக்கூடிய மகிழ் ஊர்தி(self-driving cars), கணினியோடு மக்களை விளையாடச் செய்யும் gaming industry போன்றவற்றில் ஒரு கருவிக்குத் திறம்பட பயிற்சி அளிக்க இத்தகைய முறை பயன்படுத்தப்படுகிறது. சமீபத்தில் உருவாக்கப்பட்ட 'Deepmind' என்பது இதற்கு ஒரு சிறந்த உதாரணம் ஆகும். இத்தகைய Reinforcement Learning எவ்வாறு நடைபெறுகிறது என்பது பற்றி இப்பகுதியில் விளக்கமாகக் காணலாம்.

பொதுவாக ஒருசில விஷயங்களை குழந்தைக்கு நாம் ஒருமுறை சொல்லிக்கொடுத்தால் போதும். அது சுலபமாகக் கற்றுக் கொண்டு விடும். இதனை சாதாரணமாக *learning* என்று அழைக்கலாம். ஆனால் புரிந்து கொள்வதற்குச் சற்றுக் கடினமான விஷயங்களை திரும்பத் திரும்பச் சொல்லிக்கொடுப்பதன் மூலமே ஒரு குழந்தைக்கு நாம் புரிய வைக்க முடியும். இவ்வாறு திரும்பத் திரும்பச் சொல்லிக் கொடுக்கும் முயற்சியில், ஒவ்வொரு முறையும் குழந்தையின் புரிதலுக்கு ஏற்றவாறு நாம் அடுத்தடுத்து சொல்லிக்கொடுக்க முயல்வதே *'Reinforcement Learning'* எனப்படும்.

எடுத்துக்காட்டாக ஒரு குழந்தைக்கு கடிகார முள்ளைப் பார்த்து மணி கண்டுபிடிக்க கற்றுக் கொடுக்கும் செயலை *reinforcement learning*-ல் வைத்துப் பொருத்திப் பார்ப்போம். இதில் பயன்படுத்தப்படும் முக்கிய வார்த்தைகளான *Agent, Environment, State, Action, Reward, Penalty, Policy* ஆகியவற்றை இச்செயலில் வைத்துப்

பொருத்திப் பார்த்து நாம் புரிந்து கொள்ள முயல்வோம்.

1. முதலில் ஒன்றுமே தெரியாமல் ஆரம்பித்து கடைசியில் கடிகார முள்ளைப் பார்த்து மணி சொல்ல கற்றுக்கொள்ளும் குழந்தைதான் இங்கு 'Agent' என்ற பெயரில் அழைக்கப்படுகிறது. அதாவது எதற்கு நாம் கற்றுக்கொடுக்க முயல்கிறோமோ அதுவே 'Agent' ஆகும்.

2. அடுத்து எதைப் பற்றிக் கற்றுக்கொடுக்க முயல்கிறோமோ அதுவே 'Environment' என்ற பெயரில் அழைக்கப்படுகிறது. கடிகாரம், அதில் உள்ள 60 சின்னஞ்சிறிய கோடுகள், முட்கள் உள்ள திசை, அது குறிக்கின்ற எண்கள் ஆகியவையே இங்கு environment ஆகும்.

3. Environment-க்குள் எந்த நிலையிலிருந்து நமது கற்பித்தலைத் தொடங்குகிறோமோ அதுவே *'state'* என்று அழைக்கப்படும். கடிகாரத்தில் முட்கள் இருக்கும் தற்போதைய நிலைக்கு *'state'* என்று பெயர். அதைப்பார்த்து மணியைக் கணக்கிட்டு சொல்லும் செயலுக்கு *'action'* என்று பெயர். அதாவது *environment*-ல் உள்ள *state*-க்கு ஏற்றவாறு, *agent* தன்னுடைய *action*-ஐ வெளிப்படுத்தும்.

4. Agent வெளிப்படுத்திய *action* சரியாக அமையும்பட்சத்தில் *reward*-ம், தவறாக அமையும்பட்சத்தில் *penalty*-ம் கிடைக்கும். அதாவது 'ஊக்கப்படுத்துதல்' மற்றும் 'தண்டனையளித்தல்' எனும் பொருளில் இவை இங்கு பயன்படுத்தப்படுகின்றன. கடிகார முள்ளைத் திருப்பித் திருப்பி ஒவ்வொரு முறை குழந்தையை மணி கேட்கும்போதும் சரியாகச் சொன்னால் ஒரு மிட்டாயைக் கொடுத்து ஊக்கமளிப்பது, தவறாகச் சொன்னால்

அதனிடம் உள்ள மிட்டாய்களில் ஒன்றைப் பிடுங்கிக் கொள்வது போன்ற செயல்களே இங்கு 'reward' அல்லது 'penalty' என்று அழைக்கப்படுகிறது. மிட்டாய் கிடைத்தால் தான் கற்றுக்கொண்டது சரி என்றும், மிட்டாயை இழந்தால் தான் கற்றுக்கொண்டது தவறு என்றும் குழந்தை புரிந்துகொள்ளும். எனவே அடுத்தடுத்த முறைகளில் அதிக மிட்டாய்களைப் பெறும் ஆர்வத்திலும், ஏற்கெனவே பெற்றுக்கொண்ட மிட்டாய்களை இழந்து விடக்கூடாது என்ற பயத்திலும் தனக்குக் கிடைத்த அனுபவத்தை வைத்து சரியாக மணி சொல்ல முயற்சிக்கும்.

5. சரியான மணியைக் கணக்கிட குழந்தை இதுவரை பல்வேறு வழிவகைகளைக் கண்டுபிடித்து வைத்திருக்கும். கடிகாரத்தில் உள்ள ஒவ்வொரு சின்னஞ்சிறிய கோடுகளையும் கணக்கிட்டுக் கூறுவது ஒரு வகை. 5-ம் வாய்ப்பாட்டைப் பயன்படுத்திக் கணக்கிடுவது

சற்று தேர்ந்த வகை. இதுபோன்று ஒரு விஷயத்தை நடத்தப் பயன்படுத்தும் பல்வேறு வழிவகைகளே *'strategies'* என்ற பெயரில் அழைக்கப்படுகின்றன. இதில் சிறந்த *optimized strategy*-ஐத் தேர்வு செய்து *Agent* தான் நிகழ்த்த வேண்டிய செயலை நிகழ்த்தும். இத்தகைய பல்வேறு *strategies*-ஐ உள்ளடக்கியதே *'Policy'* ஆகும்.

இதுபோன்ற *'Reinforcement Learning'* என்பது *supervised* மற்றும் *Unsupervised* என்ற இரண்டுக்கும் மத்தியில் அமையும். ஏனெனில் முதலில் *unsupervised* முறையில் ஆரம்பித்து, பின்னர் *agent*-ன் செயலைப் பொறுத்து அதனைத் தொடர்ச்சியாகத் திருத்துவதன் மூலம் *supervised* முறையில் தொடர்கிறது. கணினியுடன் நாம் விளையாடுகின்ற சீட்டுக்கட்டு விளையாட்டு,, *Chess* விளையாட்டு, கார் பந்தயம் போன்றவற்றில் நம்முடைய எதிரணியாக இருக்கும் கணினிக்குப் பயிற்றுவிக்க இத்தகைய

reinforcement தத்துவங்களே பயன்படுகின்றன. முதலில் கணினியானது அதற்குத் தெரிந்த வகையில் நம்மோடு விளையாடத் தொடங்கினாலும், பின்னர் நம்முடைய நடவடிக்கைகளுக்கு ஏற்றார் போன்று, அதனுடைய விளையாட்டுப் போக்கை மாற்றிக் கொண்டு வெற்றி பெற முயலும். எனவே அனைத்திற்கும் பொருந்தும் வகையில் இத்தத்துவங்களின் சுருக்கங்களைப் பரவலாகப் பின்வருமாறு அமைக்கலாம்.

- ➔ சூழ்நிலையைக் கண்காணித்தல் *(Observation of Environment)*

- ➔ சூழ்நிலையில் உள்ள தற்போதைய நிலைக்கு ஏற்றவாறு *Agent* தன்னுடைய அடுத்த செயலைத் தொடர்த்தல் *(Act according to the current state)*

- ➔ தன்னுடைய செயலுக்கான ஊக்க மதிப்பு அல்லது தண்டனைத் தொகையைப்

பெற்றுக் கொள்ளுதல் *(Getting reward or penalty)*

➔ *Agent* தான் பெற்றுக்கொண்ட மதிப்பிற்கு ஏற்றவாறு தன்னுடைய செயலையும், புரிதலையும் மாற்றிக் கொள்ளுதல் *(Finding a strategy from experiences)*

➔ *Agent* நிகழ்த்தும் *action* என்பது சூழ்நிலையை ஒரு நிலையிலிருந்து அடுத்த நிலைக்கு மாற்றும். இப்போது சூழ்நிலையில் ஏற்பட்டுள்ள மாற்ற நிலைக்கு ஏற்றவாறு *agent* மேற்கண்ட செயல்களை மீண்டும் மீண்டும் செய்து பல்வேறு சுழற்சிகளை உருவாக்கும். சுழற்சிகளின் முடிவில், அது பெற்றுள்ள *reward* அல்லது *penalty* மதிப்பைப் பொறுத்து சரியான வழிமுறையைக் கண்டுபிடிக்கும். *(Iterate until finding a best strategy)*

தொடக்க நிலை சுழற்சிகளில் *Agent*-க்கு தொடர்ச்சியாக ஊக்க மதிப்புகளே கிடைக்கும்பட்சத்தில் அது சரியாகப் புரிந்து கொண்டுவிட்டது என நினைத்து சுழற்சிகளை துவக்கத்திலேயே நிறுத்தி விடக்கூடாது. *(Don't be greedy)*. அதிக பட்ச சுழற்சிகளை நிகழ்த்தி அதில் தோராயமாக ஊக்கமதிப்புகள் கிடைத்துள்ளதா அல்லது தண்டனைத்தொகை கிடைத்துள்ளதா என்பதைப் பொறுத்தே சரியான தீர்மானத்தை மேற்கொள்ள வேண்டும். இந்தத் தீர்மானமானது வெறும் ஒரு நிலையைப் பொறுத்து அமையாமல், பல்வேறு காலங்களில் அமைந்த பல்வேறு நிலைகளின் தொடர்ச்சியாக அமையும். *(sequence of states according to time)*

அடுத்ததாக ஒரு ஊர்தியை தன்னிச்சையாக இயங்க வைக்க *reinforcement learning* மூலம் எவ்வாறு பயிற்சி அளிப்பது என்பதை கீழ்க்கண்ட எடுத்துக்காட்டில் காணலாம். இத்தகைய பயிற்சியின் போது பயனர்களை

எங்கிருந்து ஏற்ற வேண்டும், எங்கு இறக்க வேண்டும், இடைப்பட்ட தூரத்தை குறைந்த அளவு நேரத்தில் சரியான திசை நோக்கி எவ்வாறு கடந்து செல்வது போன்ற பல்வேறு விஷயங்களை நாம் கற்றுத்தரப் போகிறோம். ஒரு பயனரை சரியான இடத்தில் கொண்டு சேர்க்கும் செயலுக்கு அதிக அளவு வெகுமதிகளையும், ஒவ்வொரு முறை தவறான திசை நோக்கித் திரும்பும்போதும் சற்றுக் குறைந்த தண்டனை மதிப்புகளை அளித்துத் திருத்துவதன் மூலம் ஊர்திக்குக் கற்றுத் தர முடியும். இதற்குத் தகுந்த மாதிரியான தரவுகளை Gym எனும் module -க்குள் காணலாம். இதனைப் பயன்படுத்தி மேற்கண்ட விஷயங்களை செய்வதற்கான நிரலும் அதற்கான விளக்கமும் பின்வருமாறு.

https://gist.github.com/nithyadurai87/d25a4bdf226a7b5df92267e23ce8d28e

```
import gym
import numpy as np
import random

e = gym.make("Taxi-v3").env
e.render()

e.reset()
e.render()

print (e.action_space)
print (e.observation_space)

e.s = e.encode(3, 1, 2, 0) # (taxi
row, taxi column, passenger index,
destination index)
e.render()

print (e.s)
print (e.P[328])

epochs, penalties, reward = 0, 0, 0
while not reward == 20:
```

```
    action = e.action_space.sample()
    state, reward, done, info =
e.step(action) # (284, -1, False,
{'prob': 1.0})
    if reward == -10:
        penalties += 1
    epochs += 1
print("Timesteps taken:",epochs)
print("Penalties
incurred:",penalties)

q_table =
np.zeros([e.observation_space.n,
e.action_space.n])
print (q_table[328])
total_epochs, total_penalties = 0, 0
for i in range(1, 100):
    state = e.reset()
    epochs, penalties, reward, = 0,
0, 0
    done = False
    while not done:
```

```
        if random.uniform(0, 1) <
0.1:
            action =
e.action_space.sample() # Explore
action space
        else:
            action =
np.argmax(q_table[state]) # Exploit
learned values
        next_state, reward, done,
info = e.step(action)
        old_value = q_table[state,
action]
        next_max =
np.max(q_table[next_state])
        new_value = (1 - 0.1) *
old_value + 0.1 * (reward + 0.6 *
next_max)
        q_table[state, action] =
new_value
        if reward == -10:
            penalties += 1
        state = next_state
```

```
        epochs += 1
    total_penalties += penalties
    total_epochs += epochs

print (q_table[328])
print("Timesteps taken:",epochs)
print("Penalties
incurred:",penalties)
print("Average timesteps per
episode:",(total_epochs / 100))
print("Average penalties per
episode",(total_penalties / 100))
```

<u>நிரலுக்கான விளக்கம் மற்றும் வெளியீடு:</u>

1. gym.make("Taxi-v3").env எனக் கொடுப்பதன் மூலம் பயிற்சி அளிப்பதற்குத் தகுந்த சூழ்நிலை உருவாக்கப்பட்டுவிடும். இதன்மீது செயல்படும் render() மற்றும் reset() ஆகியவை சூழ்நிலையை வெளிக்காட்டுதல் மற்றும் சூழ்நிலையில் ஊர்தி உள்ள இடத்தை தோராயமாக மாற்றி அமைத்தல் போன்ற வேலைகளைச் செய்யும். இது பின்வருமாறு காணப்படும்.



- *Agent :* இதில் காணப்படும் சிறிய மஞ்சள் நிறப் பெட்டிதான் *Agent.* அதாவது இதுதான் தன்னிச்சையாக இயங்கி பயனர்களை அவரவர்குரிய இடத்தில் சென்று சேர்க்க கற்றுக் கொள்ளப்போகும் ஊர்தி.

- *Environment :* இச்சூழ்நிலையானது எளிய 5*5 வடிவ அணி போன்ற அமைப்புடன் உருவகம் செய்யப்படும். எனவே இதில்

உள்ள ஒவ்வொரு சின்ன சின்ன பகுதியையும் அணியில் பயன்படுத்தப்படும் இட மதிப்புகளை வைத்து எளிதாக அணுகலாம். அதாவது *R, G, Y, B* ஆகியவையே பயனர்களை ஏற்றி இறக்குவதற்கான இடங்கள். இவைகள் முறையே *(0,0) , (0,4) , (4,0) , (4,3)* என்று அணுகப்படும்.

- *Episode :* இதில் சூழ்நிலைக்கு ஏற்ப *Agent* செலுத்தும் *'Action'* என்பது பல்வேறு *action*-களின் தொடர்ச்சியாக அமையும். *Agent*-ன் ஒவ்வொரு செயலும் சூழ்நிலையில் மாற்றத்தை ஏற்படுத்தும். பின்னர் மாற்றப்பட்ட ஒவ்வொரு சூழ்நிலைக்கும் ஏற்ப *Agent* தனது செயலை நிகழ்த்தும். இதே முறையில் தொடர்ந்து செயல்பட்டு கடைசியில் பயனரை அவருக்குரிய இடத்தில் சென்று சேர்க்கும் செயலானது ஒரு *'episode'* என்று அழைக்கப்படுகிறது.

- *Action Space :* ஊர்தியின் செயல்பாடுகள் பின்வரும் *6* செயல்களில் ஒன்றாக அமையும். ஊர்தியை தெற்கு, வடக்கு, கிழக்கு, மேற்கு ஆகிய திசைகளில் திருப்புவது முறையே *0,1,2,3* எனவும், ஊர்தியைத் துவக்கும் செயலும், நிறுத்தும் செயலும் முறையே *4, 5* எனவும் குறிக்கப்படும். இம்மதிப்புகளே *'Action Space'* என்று அழைக்கப்படுகின்றன. *print (e.action_space)* என்பது *6* எனும் மதிப்பினை வெளிப்படுத்துவதைக் காணலாம்.

- *State Space :* சூழ்நிலையில் உள்ள ஒரு நிலைக்கு ஏற்றவாறே *Agent* தன்னுடைய செயலைத் தொடங்கும் என்று அறிவோம். மொத்தம் எத்தனை நிலைகள் இந்தச் சூழ்நிலையில் அமையலாம் என்பதே *' State Space'* எனப்படும். ஒவ்வொரு பயனரையும் உள் அழைத்துக் கொள்வது ஒரு நிலை.

*ஆகவே 5 பயனர்களுக்கு 5 நிலைகள் (R, G, Y, B ஆகிய இடங்களிலிருந்து உள் அழைத்துக் கொள்ளும் பயனர்கள் தவிர ஏற்கெனவே காருக்குள் ஒரு பயனர் இருப்பதாகக் கணக்கில் கொண்டு 5 பயனர் என்கிறோம்). அடுத்ததாக ஒவ்வொரு பயனரையும் அவரவருக்குரிய இடத்தில் சென்று சேர்ப்பது அடுத்த நிலை. 4 நிறுத்தத்திற்கான இடங்களும் 4 நிலைகள் ஆகும். கடைசியாக 5*5 வடிவமைப்பைக் கொண்ட சூழ்நிலையில் கார் திரும்பும் ஒவ்வொரு திசையும் ஒவ்வொரு நிலையைக் குறிக்கும். எனவே மொத்தம் 25 திசைகளில் கார் திரும்ப வாய்ப்பு உள்ளதால், 25 நிலைகள் உருவாகும். மேற்கூறிய 5 பயனிகள், 4 நிறுத்தங்கள், 25 திசைகள் ஆகிய அனைத்தையும் சேர்த்து state space 500 (5 * 4 * 25) என அமையும். print (e.observation_space)*

என்பது *500* எனும் மதிப்பை வெளிப்படுத்துவதைக் காணலாம்.

2. இப்போது நாமாக ஒரு நிலையை உருவாக்கிக் கொடுத்து, அதனடிப்படையில் ஊர்தியைக் கற்றுக் கொள்ளச் செய்யப் போகிறோம். *e.s = e.encode(3, 1, 2, 0)* என்பது கொடுக்கப்பட்ட சூழ்நிலையில் ஊர்த்தியை *(3,1)* எனுமிடத்தில் வைக்கும். அடுத்து தரப்பட்டுள்ள *2, 0* எனும் எண்கள் *passenger index* மற்றும் *destination index*-ஐக் குறிக்கும். *e.render()* என்பது இந்தப் புதிய நிலையை பின்வருமாறு வெளிக்காட்டும்.



3. print (e.s) என்பது *328* எனும் எண்ணை வெளிப்படுத்தும். அதாவது கொடுக்கப்பட்ட *500* நிலைகளில் இது *328*-வது நிலை என்று அர்த்தம். இந்த நிலைக்குரிய வெகுமதிக்கான அட்டவணை பின்வருமாறு அமையும் *print (e.P[328]).*

{0: [(1.0, 428, -1, False)],

1: [(1.0, 228, -1, False)],

2: [(1.0, 348, -1, False)],

3: [(1.0, 328, -1, False)],

4: [(1.0, 328, -10, False)],

5: [(1.0, 328, -10, False)]}

இதன் வடிவமைப்பு *{action1: [(probability, nextstate, reward, done)]}* எனும் அமைப்பில் இருக்கும்.

இதில் உள்ள *0,1,2,3,4,5* ஆகியவை *Agent* நிகழ்த்தும் அனைத்து வகையான செயல்களையும் குறிக்கிறது. அடுத்து உள்ள *1.0* என்பது *Agent* ஒவ்வொரு செயலையும் நிகழ்த்துவதற்கான *probability*-ஐக் குறிக்கிறது. அனைத்து செயல்களும் நடைபெறுவதற்கான வாய்ப்பு சமமாகவே இருக்குமாதலால் இதன் மதிப்பு எப்போதும் *1* என அமையும். அடுத்து உள்ள *428, 228, 348*...போன்ற எண்கள் *Agent* அக்குறிப்பிட்ட செயலை நிகழ்த்தினால் சென்றடையும் அடுத்த நிலைக்கான எண்ணைக் குறிக்கும். எனவே தான் *3* எனுமிடத்தில் ஏற்கெனவே ஊர்தி வைக்கப்பட்டுள்ளதாலும், *4, 5* ஆகியவை பயனரை ஏற்றி இறக்குவதற்கான செயல்கள் மட்டுமே என்பதாலும் *328* நிலையில் மாற்றம் ஏற்படாமல் அதே எண்ணை வெளிப்படுத்தியுள்ளது. அடுத்து உள்ள *-1, -10* ஆகியவை ஒவ்வொரு செயலுக்குமான வெகுமதி மதிப்புகள் ஆகும். கடைசி செயலுக்கான வெகுமதி மதிப்பு மட்டும், அதனையடுத்து உள்ள மதிப்பு *True* என மாறும்போது *20* என அமையும். அதாவது

கடைசியில் உள்ள *False* என்பது பயனரை இன்னும் அவருக்கான இடத்தில் இறக்கிவிடவில்லை என்பதைக் குறிக்கிறது. அவரை வெற்றிகரமாக இறக்கிவிட்டபின், இம்மதிப்பு *True* எனவும், கடைசி செயலுக்கான வெகுமதி மதிப்பு *20* எனவும் மாறிவிடும். இதுவே ஒரு *'Episode'* நிறைவுற்றத்தைக் குறிக்கும்.

4. இப்போது *RL*-க்கான எந்த ஒரு *algorithm*-ஐயும் பின்பற்றாமல், *random*-ஆக ஊர்தியை அடுத்தடுத்த திசை நோக்கி நகர வைப்பதற்கான *loop* பின்வருமாறு அமையும். இது கடைசி செயலான *drop-off* க்குக் கிடைக்கின்ற வெகுமதி மதிப்பு *20* என அமையாத வரை ஊர்தியை தொடர்ந்து *destination*-ஐ நோக்கிப் பயணிக்க வைக்கின்ற *while not loop* ஆகும். *e.action_space.sample()* என்பது மொத்த செயல்களில் இருந்து *random*-ஆக ஒரு செயலை எடுத்து நிகழ்த்தும். *e.step(action)* என்பது

அச்செயலை நிகழ்த்துவதால் வருகின்ற அடுத்த நிலைக்கான மதிப்பு, அச்செயலுக்குப் பெறுகின்ற வெகுமதி மதிப்பு, அச்செயலால் காரியம் முற்றுப்பெற்றதா இல்லையா என்பதை *(284, -1, False, {'prob': 1.0})* என்ற வடிவில் வெளிப்படுத்தும். இதில் முதல் மூன்று மதிப்புகளே *state, reward, done* போன்ற *variables*-ல் சேமிக்கப்படுகின்றன. ஒவ்வொரு முறை தவறாகப் பயனரை இறக்கிவிடும்போதும், *penalty* மதிப்புடன் *1* கூட்டப்படுகிறது. கடைசியாகப் பயனரை அவருக்குரிய சரியான இடத்தில் இறக்கிவிடும்போது வெகுமதி மதிப்பு *20* என மாறி *loop*-ஐ விட்டு வெளியேறுகிறது. பின்னர் ஒரு *episode* ஐ முடிப்பதற்கு மொத்தம் எத்தனை முறை நகர்ந்துள்ளது என்பதையும், எத்தனை முறை பயனரை தவறான இடத்தில் இறக்கி *penalties* வாங்கியுள்ளது என்பதையும் வெளிப்படுத்துகிறது.

```
epochs, penalties, reward = 0, 0, 0

while not reward == 20:

    action = e.action_space.sample()

    state, reward, done, info = e.step(action)

    if reward == -10:

        penalties += 1

    epochs += 1
```

Timesteps taken: 1485

Penalties incurred: 473

5. Q-Learning Algorithm: மேற்கண்ட படியில் மொத்தம் *1485* முறை நகர்ந்துள்ளதையும், *473* முறை தவறாக இறக்கி *penalties* வாங்கியுள்ளதையும் காணலாம். இப்போது *Q-Learning* எனும் *RL*-க்கான *algorithm*-ஐப் பயன்படுத்தும்போது, இம்மதிப்புகள் குறைவதைக் காணலாம். ஒரு *Agent* சூழ்நிலையில் தன்னுடைய செயலுக்கு ஏற்றபடி கிடைக்கின்ற வெகுமதி மதிப்புகளை நினைவில் வைத்துக் கொண்டு அதன்படி அடுத்தடுத்து செயல்படுவதன் மூலம் தவறுகளைக் குறைக்க முடியும். இதுபோன்ற *memory*-ஐ *Agent*-க்கு வழங்க இந்த *algorithm* பயன்படுத்துவதே *Q-table* ஆகும். இதில் ஒவ்வொரு நிலைக்குமான பல்வேறு செயல்களின் தாக்கங்கள் எந்த அளவுக்கு அமைகின்றன எனும் மதிப்பு சேமிக்கப்படுகிறது. இதுவே *Q* மதிப்பு ஆகும்.

இது ஒவ்வொரு செயலின் தரத்தையும் உணர்த்தக் கூடியது.

Q-table என்பது *State * Action* எனும் அணி வடிவ அமைப்பில் அமையும். அதாவது மொத்தம் உள்ள 500 நிலைகளும் *states* ஆகவும், ஒவ்வொரு நிலைக்குமான 6 செயல்களின் மதிப்புகளும் *columns*-ஆகவும் அமையும். இதனடிப்படையில்தான் *Q* மதிப்புகள் சேமிக்கப்படுகின்றன. ஒவ்வொரு நிலைக்கும் எந்த செயலுக்கான *Q* மதிப்பு அதிகமாக உள்ளதோ அதுவே பொருத்தமான செயலாகக் கணக்கில் கொள்ளப்படும் .

6. நம்முடைய நிரலில் முதலில் 0 மதிப்பினைப் பெற்று விளங்கும் *Q-table* உருவாக்கப்படுகிறது. இதன் *dimension 500 * 6* என இருக்கும்படி பின்வருமாறு அமைக்கப்படுகிறது. இப்போது 328-வது நிலைக்கான அனைத்து செயல்களும் 0 எனும் துவக்க மதிப்புகளைப் பெற்றுள்ளத்தைக் காணலாம்.

```
q_table = np.zeros([e.observation_space.n,
e.action_space.n])

print (q_table[328])

[0. 0. 0. 0. 0. 0.]
```

இம்மதிப்புகளை *update* செய்யும் நிகழ்வானது *100* முறை *for loop* மூலம் நிகழ்த்தப்படுகிறது. அதாவது *100* முறை பயனர்களைச் சரியாகக் கொண்டு சேர்க்கப் பயன்படுத்திய *Q* மதிப்புகள் ஒவ்வொரு முறையும் இதில் *update* செய்யப்படும். ஒவ்வொரு செயலும் அதற்கு முன்னர் அதிக *Q* மதிப்பைப் பெற்று அமைந்த செயலின்படி வழிநடத்தப்படும். இவ்வாறாக *100 episodes* நிறைவுற்ற பின்னர் அதே நிலைக்கான *Q* மதிப்பு பின்வருமாறு அமைவதைக் காணலாம்.

```
[-1.12496344 -1.12745359 -1.09896023 -1.0872184  -
2.80419013 -2.78076376]
```

இம்மதிப்புகளை வைத்துப் பார்க்கும்போது அதிக மதிப்பைப் *(-1.0872184)*பெற்று விளங்கும் *3*-வது செயலை இந்நிலைக்குப் பொருத்தமான செயலாக எடுத்துக் கொள்ளலாம். ஆனால் இதுபோன்ற குறைந்த *episodes*-ஐ நிகழ்த்தி ஒரு முடிவுக்கு வரக்கூடாது. இதையே *learning by greedy* என்போம். குறைந்த பட்சம் பல லட்சம் சுழற்சிகளையாவது நிகழ்த்திய பின்னரே எச்செயல் பொருத்தமானதாக அமையும் என்ற முடிவுக்கு நம்மால் வர முடியும். இதன் பின்னர் கடைசி *100*-வது *episode*-ல் ஊர்தியின் நகர்வு எண்ணிக்கையும், வாங்கிய *penalty* மதிப்பும் வெளிப்படுத்தப்பட்டுள்ளது. மேலும் சராசரியாக ஒரு *episode*-க்கு இத்தகைய மதிப்புகள் என்னென்ன என்பதும் வெளிப்படுத்தப்பட்டுள்ளது. இம்மதிப்புகள் இதற்கு முன்னர் வெளிப்படுத்திய மதிப்புகளை விடக் குறைவாக அமைவதைக் காணலாம். இதுவே ஊர்தி *Q-algorithm* மூலம் சிறப்பாக செயல்படக் கற்றுக் கொண்டு விட்டதை உறுதிப்படுத்தும்.

Timesteps taken: 331

Penalties incurred: 16

Average timesteps per episode: 452.2

Average penalties per episode 24.67

7. ஒவ்வொரு சுழற்சியிலும் Q-Table ஐ மேம்படுத்தும் நிகழ்வானது பின்வரும் சமன்பாட்டின் படி நடைபெறும்.

*new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)*

இதில் alpha, gamma ஆகியவை hyperparameters . Alpha என்பது ஒவ்வொரு சுழற்சியிலும் பயன்படுத்தப்படும் learning rate-ஐக் குறிக்கும். இது 0.1 என அமைந்துள்ளது .

gamma என்பது 0 முதல் 1 வரை அமைந்த மதிப்பைப் பெற்றிருக்கும். இங்கு 0.6 என

அமைந்துள்ளது . இதன் மதிப்பை *0* -க்கு நெருக்கத்தில் அமைக்கும்போது, *quick rewards*-ஐ வைத்து அதிக பொருத்தமான செயலைத் தேர்ந்தெடுக்கும். *1*-க்கு அருகில் அமைக்கும்போது, *long-term rewards*-ஐ வைத்து தேர்ந்தெடுக்கும்.

மேலும் *epsilon* எனும் ஒரு *parameter* இங்கு *overfitting*-ஐத் தடுக்கப் பயன்படுகிறது. *for loop*-ன் துவக்கத்தில், இந்த மதிப்பைப் பொறுத்தே ஆக ஒரு செயலை *random*-ஆகத் தேர்ந்தெடுக்க வேண்டுமா அல்லது அதிக *Q* மதிப்பைப் பெற்ற ஒரு செயலைத் தேர்ந்தெடுக்க வேண்டுமா என்பதை முடிவு செய்யும். ஏனெனில் ஒவ்வொரு முறையும் அதிக *Q* மதிப்பைப் பெற்ற செயல்களே திரும்பத் திரும்பத் தேர்ந்தெடுக்கப்படும்போது *over fitting* ஏற்பட வாய்ப்பு உள்ளது. ஆகவே இதனைத் தவிர்க்க *epsilon*-க்கு ஒரு மதிப்பினை வரையறுத்து அதனடிப்படையில் சிலசமயம் *random*-ஆகவும், சில சமயம் அதிக *Q* மதிப்பு கொண்ட செயலும் தேர்ந்தெடுக்கப்படுகிறது.

19 முடிவுரை

இத்துடன் Deep Learning முடிவடைந்து விடவில்.லை. AI, Neural Networks என்று பல்வேறு புதுமைகள் கணினி உலகில் நடந்து வருகின்றன. அவற்றை இணையத்தில் கிடைக்கும் பாடங்கள், வலைப்பதிவுகள், StackOverFlow போன்ற தளங்கள், காணொளிகள் வழியே தொடர்ந்து கற்று வர வேண்டுகிறேன்.

நன்றி

20 ஆசிரியரின் பிற மின்னூல்கள்

---

http://freetamilebooks.com/authors/nithyaduraisamy/


எளிய தமிழில்
HTML
எளிய தமிழில்
MySQL
து. நித்யா
கணியம் வெளியீடு
எளிய தமிழில்
MySQL
து நித்யா
2
எளிய தமிழில்
GNU
Linux
பாகம் - 1
து. நித்யா
கணியம் வெளியீடு
எளிய தமிழில்
Se
Selenium
து.நித்யா
எளிய தமிழில்
BIG DATA
து. நித்யா
எளிய தமிழில்
JavaScript
JS
து.நித்யா
எளிய தமிழில்
GNU
Linux
பாகம் - 2
து. நித்யா
கணியம் வெளியீடு
எளிய தமிழில்
CSS
CSS
து. நித்யா

21 கணியம் பற்றி

இலக்குகள்

கட்டற்ற கணிநுட்பத்தின் எளிய விஷயங்கள் தொடங்கி அதிநுட்பமான அம்சங்கள் வரை அறிந்திட விழையும் எவருக்கும் தேவையான தகவல்களை தொடர்ச்சியாகத் தரும் தளமாய் உருபெறுவது.

- *உரை, ஒலி, ஒளி என பல்லூடக வகைகளிலும் விவரங்களை தருவது.*
- *இத்துறையின் நிகழ்வுகளை எடுத்துரைப்பது.*

- எவரும் பங்களிக்க ஏதுவாய் யாவருக்குமான நெறியில் விவரங்களை வழங்குவது.

- அச்சு வடிவிலும், புத்தகங்களாகவும், வட்டுக்களாகவும் விவரங்களை வெளியிடுவது.

பங்களிக்க

- விருப்பமுள்ள எவரும் பங்களிக்கலாம்.

- கட்டற்ற கணிநுட்பம் சார்ந்த விஷயமாக இருத்தல் வேண்டும்.

- பங்களிக்கத் தொடங்கும் முன்னர் கணியத்திற்கு உங்களுடைய பதிப்புரிமத்தை அளிக்க எதிர்பார்க்கப்படுகிறீர்கள்.

- **editor@kaniyam.com**

  முகவரிக்கு கீழ்க்கண்ட

  விவரங்களடங்கிய மடலொன்றை

  உறுதிமொழியாய் அளித்துவிட்டு

  யாரும் பங்களிக்கத் தொடங்கலாம்.

  - ***மடலின் பொருள்:*** *பதிப்புரிமம் அளிப்பு*

  - ***மடல் உள்ளடக்கம்***

    - *என்னால் கணியத்திற்காக அனுப்பப்படும் படைப்புகள் அனைத்தும் கணியத்திற்காக முதன்முதலாய்*

படைக்கப்பட்டதாக உறுதியளிக்கிறேன்.

- இதன்பொருட்டு எனக்கிருக்கக்கூடிய பதிப்புரிமத்தினை கணியத்திற்கு வழங்குகிறேன்.
- உங்களுடயை முழுப்பெயர், தேதி.

- தாங்கள் பங்களிக்க விரும்பும் ஒரு பகுதியில் வேறொருவர் ஏற்கனவே பங்களித்து வருகிறார் எனின் அவருடன் இணைந்து பணியாற்ற முனையவும்.
- கட்டுரைகள் மொழிபெயர்ப்புகளாகவும், விஷயமறிந்த ஒருவர் சொல்லக் கேட்டு கற்று இயற்றப்பட்டவையாகவும் இருக்கலாம்.

- படைப்புகள் தொடர்களாகவும் இருக்கலாம்.

- தொழில் நுட்பம், கொள்கை விளக்கம், பிரச்சாரம், கதை, கேலிச்சித்திரம், நையாண்டி எனப் பலசுவைகளிலும் இத்துறைக்கு பொருந்தும்படியான ஆக்கங்களாக இருக்கலாம்.

- தங்களுக்கு இயல்பான எந்தவொரு நடையிலும் எழுதலாம்.

- தங்களது படைப்புகளை எளியதொரு உரை ஆவணமாக editor@kaniyam.com முகவரிக்குஅனுப்பிவைக்கவும்.

- தள பராமரிப்பு, ஆதரவளித்தல் உள்ளிட்ட ஏனைய விதங்களிலும் பங்களிக்கலாம்.

- ஐயங்களிருப்பின் editor@kaniyam.com மடலியற்றவும்.

விண்ணப்பங்கள்

- கணித் தொழில்நுட்பத்தை அறிய விழையும் மக்களுக்காக மேற்கொள்ளப்படும் முயற்சியாகும் இது.

- இதில் பங்களிக்க தாங்கள் அதிநுட்ப ஆற்றல் வாய்ந்தவராக இருக்க வேண்டும் என்ற கட்டாயமில்லை.

- தங்களுக்கு தெரிந்த விஷயத்தை இயன்ற எளிய முறையில் எடுத்துரைக்க ஆர்வம் இருந்தால் போதும்.

- இதன் வளர்ச்சி நம் ஒவ்வொருவரின் கையிலுமே உள்ளது.

- குறைகளிலிருப்பின் முறையாக தெரியப்படுத்தி முன்னேற்றத்திற்கு வழி வகுக்கவும்.

வெளியீட்டு விவரம்

ஆசிரியர்: த. சீனிவாசன் –
editor@kaniyam.com +91 98417
95468

*கட்டுரைகளில் வெளிப்படுத்தப்படும்
கருத்துக்கள் கட்டுரையாசிரியருக்கே உரியன*

2.1 தொலை நோக்கு – *Vision*

தமிழ் மொழி மற்றும் இனக்குழுக்கள் சார்ந்த மெய்நிகர்வளங்கள், கருவிகள் மற்றும் அறிவுத்தொகுதிகள், அனைவருக்கும் கட்டற்ற அணுக்கத்தில் கிடைக்கும் சூழல்

2.2 பணி இலக்கு – *Mission*

அறிவியல் மற்றும் சமூகப் பொருளாதார வளர்ச்சிக்கு ஒப்ப, தமிழ் மொழியின் பயன்பாடு வளர்வதை உறுதிப்படுத்துவதும், அனைத்து அறிவுத் தொகுதிகளும், வளங்களும் கட்டற்ற அணுக்கத்தில் அனைவருக்கும் கிடைக்கச்செய்தலும்.

2.3 தற்போதைய செயல்கள்

- கணியம் மின்னிதழ் – kaniyam.com
- கிரியேட்டிவ் காமன்சு உரிமையில் இலவச தமிழ் மின்னூல்கள் – FreeTamilEbooks.com

2.4 கட்டற்ற மென்பொருட்கள்

- உரை ஒலி மாற்றி – Text to Speech
- எழுத்துணரி – Optical Character Recognition
- விக்கிமூலத்துக்கான எழுத்துணரி

- மின்னூல்கள் கிண்டில் கருவிக்கு அனுப்புதல் – Send2Kindle
- விக்கிப்பீடியாவிற்கான சிறு கருவிகள்
- மின்னூல்கள் உருவாக்கும் கருவி
- உரை ஒலி மாற்றி – இணைய செயலி
- சங்க இலக்கியம் – ஆன்டிராய்டு செயலி
- FreeTamilEbooks – ஆன்டிராய்டு செயலி
- FreeTamilEbooks – ஐஓஎஸ் செயலி

- WikisourceEbooksReport இந்திய மொழிகளுக்ககான விக்கிமூலம் மின்னூல்கள் பதிவிறக்கப் பட்டியல்
- FreeTamilEbooks.com – Download counter மின்னூல்கள் பதிவிறக்கப் பட்டியல்

.5 அடுத்த திட்டங்கள்/மென்பொருட்கள்

- விக்கி மூலத்தில் உள்ள மின்னூல்களை பகுதிநேர/முழு நேரப் பணியாளர்கள் மூலம் விரைந்து பிழை திருத்துதல்

- முழு நேர நிரலரை பணியமர்த்தி பல்வேறு கட்டற்ற மென்பொருட்கள் உருவாக்குதல்

- *தமிழ் NLP க்கான பயிற்சிப் பட்டறைகள் நடத்துதல்*

- *கணியம் வாசகர் வட்டம் உருவாக்குதல்*

- *கட்டற்ற மென்பொருட்கள், கிரியேட்டிவ் காமன்சு உரிமையில் வளங்களை உருவாக்குபவர்களைக் கண்டறிந்து ஊக்குவித்தல்*
- *கணியம் இதழில் அதிக பங்களிப்பாளர்களை உருவாக்குதல், பயிற்சி அளித்தல்*

- *மின்னூலாக்கத்துக்கு ஒரு இணையதள செயலி*

- *எழுத்துணரிக்கு ஒரு இணையதள செயலி*

- தமிழ் ஒலியோடைகள் உருவாக்கி வெளியிடுதல்
- OpenStreetMap.org ல் உள்ள இடம், தெரு, ஊர் பெயர்களை தமிழாக்கம் செய்தல்
- தமிழ்நாடு முழுவதையும் OpenStreetMap.org ல் வரைதல்
- குழந்தைக் கதைகளை ஒலி வடிவில் வழங்குதல்
- Ta.wiktionary.org ஐ ஒழுங்குபடுத்தி API க்கு தோதாக மாற்றுதல்
- Ta.wiktionary.org க்காக ஒலிப்பதிவு செய்யும் செயலி உருவாக்குதல்

- *தமிழ் எழுத்துப் பிழைத்திருத்தி உருவாக்குதல்*

- *தமிழ் வேர்ச்சொல் காணும் கருவி உருவாக்குதல்*

- *எல்லா* FreeTamilEbooks.com *மின்னூல்களையும்* Google Play Books, GoodReads.com *ல் ஏற்றுதல்*

- *தமிழ் தட்டச்சு கற்க இணைய செயலி உருவாக்குதல்*
- *தமிழ் எழுதவும் படிக்கவும் கற்ற இணைய செயலி உருவாக்குதல்* (aamozish.com/Course_preface *போல*)

மேற்கண்ட திட்டங்கள், மென்பொருட்களை உருவாக்கி செயல்படுத்த உங்கள் அனைவரின் ஆதரவும் தேவை. உங்களால் எவ்வாறேனும் பங்களிக்க இயலும் எனில் உங்கள் விவரங்களை kaniyamfoundation@gmail.com க்கு மின்னஞ்சல் அனுப்புங்கள்.

2.6 வெளிப்படைத்தன்மை

கணியம் அறக்கட்டளையின் செயல்கள், திட்டங்கள், மென்பொருட்கள் யாவும் அனைவருக்கும் பொதுவானதாகவும், 100% வெளிப்படைத்தன்மையுடனும் இருக்கும். இந்த இணைப்பில் செயல்களையும், இந்த இணைப்பில் மாத அறிக்கை, வரவு செலவு விவரங்களுடனும் காணலாம்.

கணியம் அறக்கட்டளையில் உருவாக்கப்படும் மென்பொருட்கள் யாவும் கட்டற்ற மென்பொருட்களாக மூல நிரலுடன், GNU GPL, Apache, BSD, MIT, Mozilla ஆகிய உரிமைகளில் ஒன்றாக வெளியிடப்படும். உருவாக்கப்படும் பிற வளங்கள், புகைப்படங்கள், ஒலிக்கோப்புகள், காணொளிகள், மின்னூல்கள், கட்டுரைகள் யாவும் யாவரும் பகிரும், பயன்படுத்தும் வகையில் கிரியேட்டிவ் காமன்சு உரிமையில் இருக்கும்.

2.7 நன்கொடை

உங்கள் நன்கொடைகள் தமிழுக்கான கட்டற்ற வளங்களை உருவாக்கும் செயல்களை சிறந்த வகையில் விரைந்து செய்ய ஊக்குவிக்கும்.

பின்வரும் வங்கிக் கணக்கில் உங்கள் நன்கொடைகளை அனுப்பி, உடனே விவரங்களை kaniyamfoundation@gmail.com *க்கு மின்னஞ்சல் அனுப்புங்கள்.*

Kaniyam Foundation
Account Number : 606 1010 100 502 79

Union Bank Of India
West Tambaram, Chennai

IFSC – UBIN0560618

Account Type : Current Account

2.8 UPI செயலிகளுக்கான QR Code

குறிப்பு: சில UPI செயலிகளில் இந்த QR Code வேலை செய்யாமல் போகலாம். அச்சமயம் மேலே உள்ள வங்கிக் கணக்கு எண், IFSC code ஐ பயன்படுத்தவும்.

Note: Sometimes UPI does not work properly, in that case kindly use Account number and IFSC code for internet banking.

BHIM UPI Payments Accepted at

Kaniyam Foundation

Account Number : 606101010050279, IFSC Code: UBIN0560618

Scan and Pay using any UPI supported Apps

iMobile	BHIM	PhonePe	PayTM	SBI Pay	Google Tez
RBL Pay	DBS	PNB UPI	Yes Pay	AXIS Pay	Chiller
Union UPI	HDFC	Baroda Pay	Indus Pay	BOI UPI	Maha UPI

Generated By : https://nsisodiya.github.io/Universal-UPI-QR-Code-Generator